PARALLEL GLOBAL OPTIMIZATION ALGORITHMS ON COMPUTATIONALLY EXPENSIVE GROUNDWATER PROBLEMS

A Dissertation

Presented to the Faculty of the Graduate School

of Cornell University

in Partial Fulfillment of the Requirements for the Degree of

Doctor of Philosophy

by

Min Pang

August 2017

PARALLEL GLOBAL OPTIMIZATION ALGORITHMS ON COMPUTATIONALLY EXPENSIVE GROUNDWATER PROBLEMS

Min Pang, Ph.D.

Cornell University 2017

This dissertation explores the development and implementation of Parallel Surrogate-based Optimization Algorithms which are designed specifically for problems with multiple local optima and require a large computation burden. With the help of multiple variants of Surrogate-based Optimization algorithms, different computationally expensive groundwater problems can be solved within a limited computation budget.

In the first part of the dissertation, we examine parallelism performance of Parallel Stochastic Radial Basis Function (p-SRBF) Algorithm on groundwater management problems. SRBF uses Radial basis functions as a surrogate surface to assist the search for an optimal solution. In its revised Parallel version, p-SRBF is able to reach super-linear speed-up and reduce the computation budget by a large factor in the two problems we worked on. Both of the problems deal with management of pump and treat systems for remediation on contaminated Superfund sites. The analysis also shows that p-SRBF performs the best among all three popular parallel optimization algorithms compared, including Parallel Genetic Algorithm, Parallel NOMAD and APPSPACK.

The second part concentrates in the area of constraint dealing strategy. The problem is based on a real world model simulating land subsidence induced by overdraft of groundwater. In order to help decision makers plan groundwater exploitation in an efficient way given the land subsidence and demand occuring in

the region, we develop a parallel version of DYSOC which incorporates strategy for consideration of expensive constraints in the process of exploring optimal objective function. DYSOC is developed based on DYCORS which is a variant of SRBF for problems with high dimensions. Three different scenarios are introduced, each describing a situation of pumping in different considerations, including maximizing extraction, prioritizing environmental protection or exploiting deep pumping. Results indicates that DYSOC is efficient compared to p-SRBF.

The third part is dedicated to an exploration of the asynchronous paradigm of Parallel Surrogate-based Optimization algorithms using the PySOT toolbox in a calibration problem of groundwater flow and transport model. To efficiently incorporate the asynchronous version, we develop an early truncation strategy in each simulation so that for parameter sets which may lead to a bad solution, only partial simulation is spent to save the computing budget. Various coupling strategies have been tested on two different sites in the Umatilla problem. The asynchronous version algorithm with coupled Early Truncation Strategy shows a consistent and robust performance. In addition, in comparison with algorithms SCE-UA and APPSPACK, our coupled paradigm also has an better performance.

BIOGRAPHICAL SKETCH

Min Pang grew up in Nanjing, Jiangsu Province, China and graduated from Nanjing Jinling High School in 2006. She graduated from Hohai University in 2011 with Bachelor of Science degrees in Thermodynamic Engineering. During the undergraduate study, she participated in exchange program of Hohai University and University of Science and Technology in Lille (Lille1) France, for two and half years and received Bachelor of Science degree in Civil Engineering from Lille1 in France.

In 2011, she began her study at Cornell University and was awarded Master of Engineering degrees in Environmental and Water Resources systems Engineering in 2012. From then on, she worked under supervision of Professor Christine Shoemaker with a focus on implementation of Parallel Surrogate-based Optimization algorithm in computationally expensive real-world simulation models by using Super-computer systems. She had the chance to work at Peking University in China in 2014 on project of Controlling Land subsidence in Hang-Jia-Hu areas under the guidance of Professor Chunmiao Zheng for one semester. And she worked with Professor Christine Shoemaker at National University of Singapore in 2015 and 2016 for studies on asynchronous Parallel Optimization algorithms.

This document is dedicated to my family and friends.

ACKNOWLEDGEMENTS

I would like to express the deepest appreciation to my committee chair Professor Christine Shoemaker for her academic supervise, devoted time and financial support all through my PhD studies. Her wide horizons and research interests in scientific studies broadened my knowledge in the area of mathematics, computer science and operational research. In addition, her research passion inspired my interest in making theoretical methods applicable to complicated real world problems. During my PhD study, she provided me valuable connection to work in joint project with Professor Chunmiao Zheng. This experience expanded my understanding in groundwater modeling and hydrological system development. Moreover, I have had the honor work with Professor David Bindel at Cornell, who help me get exposed to deep understanding of mathematical algorithms and the method of developing toolbox for new algorithms .

I would also like to thank my committee member Professor Philip Liu and Professor Huseyin Topaloglu for their time and help in my PhD studies, and I would like to acknowledge high-performance computing support from Yellowstone (ark:/85065/d7wd3xhc) provided by NCAR's Computational and Information Systems Laboratory, sponsored by the National Science Foundation. All analysis have been based on experiments in this system. In addition, thanks to my friends and colleagues Ying Wan, SueNee Tan, Juliane Mueller, David Eriksson, Tamioor Akhtar, Xin Yu from our EWRS (water resources and system engineering) group Cornell for their valuable assistance and support, and also thanks to Cornell CEE department for financial support as Teaching Assistant during my studies.

Last but not least, I would like to my family for their unconditional love and support throughout the PhD studies and my life in general.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1

INTRODUCTION

Many problems in engineering involve complex numerical simulations of physical systems, which require long execution time. Often the inner components of the models are hardly available for inspection and in many cases, the mathematical characteristics, like derivative information, are not available. This type of simulation system can be regarded as a "black-box" for which the models are viewed only in terms of their inputs and outputs. Complex groundwater models which involve solving large systems of partial differential equations can be regarded as this type of "black-box" functions in order to solve for problems such as management of the system, simulator calibration, and monitoring network design. Their solving procedures are based on a Simulation and Optimization framework, in which the simulation model is coupled with a mathematical optimization algorithm. In the framework, we interchange simulation evaluations and inputs chosen from optimization to find an optimal solution. However, the optimal solution often involves study of simulation models in response to various input stimuli, therefore we need to evaluate a large number of the simulation models. The problem is that many sequential simulations are unaffordable if there is a great computational demand of the black box simulation function. Therefore in dealing with such problems, an efficient parallel optimization algorithm which evaluates the simulation models simultaneously can greatly reduce the computational time necessity.

The parallelism is realized by making use of the multi-core processing architecture in the computing system. As parallel computer architectures have become easily accessible, different parallel algorithms have been actively developed. The early parallel optimization algorithms build on mathematical knowledge lying un-

der gradient method such as Newton's method, least square minimization and largely sequential techniques. In spite of their rapid convergence to local optimal, these methods are not good at finding global optimal and have few independent tasks that can be performed in parallel. Later on, direct search methods were developed, which have potentially greater concurrency. In addition, heuristic methods such as evolutionary algorithm are also becoming popular, which can also be implemented in parallel. For these types of methods, large number of simulations are still required. In that case, massive computing cores are in need given a limited time frame, which can cause low efficiency performed by the parallel algorithm. In this study, we use the efficient parallel optimization methods based on surrogate-surface, which both reduce the number of simulations necessary as well as the computing time as a whole. It is a derivative-free method and is targeted to global optimization problems which have multiple local optima. The efficiency of the basic parallel algorithm Stochastic RBF is discussed in its application on groundwater remediation problems (Chapter 2). This real world application is based on systems that couple a groundwater flow and transport model in two superfund contamination sites, aiming at reducing the fixed or variable cost for reaching a certain remexdiating level. We demonstrate that Parallel Stochastic RBF outperforms three alternative parallel algorithms when applied on both contamination clean-up models. Chapter 3 addresses the issue in a land-surface subsidence control problem associated with overdraft of groundwater, where optimization has rarely been discussed for such an integrated model. For this problem involving a large number of controlling wells, we use the surrogate-based parallel algorithm DYCORS that aims at solving high-dimensional problems, and examine different constraint-dealing strategies concerning land subsidence limitation and required pumping of groundwater. Chapter 4 discusses the asynchronous parallelism strat-

egy, which is suitable for simulations with varying computation time. This applies to a parameter calibration problem in which early truncation techniques can be conducted in order to reduce computation time.

CHAPTER 2

COMPARISON OF PARALLEL GLOBAL OPTIMIZATION ALGORITHMS ON COMPUTATIONALLY EXPENSIVE GROUNDWATER REMEDIATION DESIGNS

2.1 Introduction

Groundwater occurs almost everywhere beneath the land surface, but demand for it is growing worldwide. However, groundwater contamination problems induced by human activities place constraints on groundwater availability and also pose a threat to human health and the surrounding environment [3]. Many technologies have been developed for alleviating these problems. Among them, pump and treat system design (P&T) is an effective method for pollutant removal and is the most widely used method in many regions.

However, without an optimal remediation plan, the financial cost associated with the implementation of a P&T system can be very large [51]. It has been shown that the Simulation/optimizatinon (S/O) framework, which integrates a mathematical optimization algorithm with a physical-based model (i.e. groundwater flow and transport model) can provide an optimal pumping strategy with a huge cost saving [77]. However, due to the complexity of the real field situation, simulation based on a large field-scale model is usually computational expensive. Therefore, an efficient optimization is then essential to find an optimal solution. Today, the criteria "effective" defines not only the optimization strategy itself, but also its ability to harness the power of the high-performance computing (HPC) resources.

Along with the rapid development of HPC systems, many optimization algorithms adopting parallelization approach are executed on clusters to reduce execution time. Among them, the parallel evolutionary algorithm is a popular strategy that has been applied in many groundwater problems. The Parallel real-valued genetic algorithm (GA) was adopted for bioremediation optimization of TCE-contaimnated groundwater [21]. Yan and Minsker (2006) developed a hybrid algorithm employing both Neural Network and GA and used the parallelism technique to obtain an optimal groundwater remediation design [75]. However, there was little discussion on the efficiency of the parallelism in this approach in either studies. Yang et al. (2013) discussed speedup and efficiency of a new parallel hybrid multi-objective algorithm for groundwater remediation design [76], and in other applications, Sayeed et al. (2005) measured the speedup of implementing a parallel optimization based on GA and several local searches in a hybrid mode on groundwater inverse problem [62]. Mirghani et al. (2009) assessed the speedup and scalability of utilizing a parallel evolutionary search algorithm in an S/O approach for solving groundwater source identification problems [49]. In these studies, speedup of the algorithm is evaluated in terms of a fixed number of simulations. [54] demonstrated that parallel search performance on speedup measured by computational time to achieve same amount of tasks using different number of cores cannot reflect the exploiting ability of the evolutionary algorithm, and speedup and efficiency related to achieving the same result accuracy are better measures to evaluate the algorithm. However, the evolutionary algorithm requires hundreds of thousands of simulations to be computed in order to obtain the optimal solution to some accuracy level, which limits its application on computationally demanding problem when a single simulation may take hours. The computational burden of using these parallel algorithms on such problems can be tremendous and hard to

realize.

Our test problems are two computationally expensive groundwater remediation problems. The first one is at Umatilla Aquifer which takes about 1.5 minutes for a single simulation of the hydrological model. Various studies of different optimization algorithms have been tested on Umatilla ([65],[80],[77],[40],[68],[12]) and the majority of the optimization strategies are based on evolutionary algorithm. The second problem is Blaine Aquifer for which each simulation takes around 30 minutes. Fewer studies have been dedicated to the Blaine problem due to its computationally intensive nature. ([77],[41], [27]).

In our study, we applied a new Parallel algorithm: Parallel Stochastic Radial Basis Function (p-SRBF) [58]. Our implementation has a slight difference in restart criterion from algorithm ParLMSRS in [58]. It is a derivative-free parallel optimization based on the Surrogate Surface Method which has not yet been implemented in any groundwater design work. The Parallel SRBF algorithm in Python calls the computationally expensive groundwater model MODFLOW-MT3D in Fortran and they form an integrated model as a whole. We implement this model on NCAR super-computer system and conduct a performance analysis in terms of the parallelism efficiency in computation and robustness of the stochastic search. In addition, we compare with other derivative-free optimization techniques chosen from three classical parallel algorithms such as Mesh-based methods and population-based methods, which have been applied in many groundwater optimization problems.

The description of all four algorithms is given in Section 2.2 including their individual implementations in our experiment. In Section 2.3, a brief background information and formulation of our test problems are provided, followed by the

computational set-up specification in Section 2.4. Detailed results and discussions are in Section 2.5.

2.2 Parallel Optimization Algorithms

2.2.1 Parallel Stochastic Radial Basis Function (p-SRBF)

Parallel Stochastic Radial Basis function (p-SRBF) is our new modification of the parallel version of a global optimization method developed by [58] called as ParLM-SRBF. It is a derivative-free algorithm that is suitable for optimizing multimodal black-box functions. The RBF method is adopted as a surrogate surface that guides the search in order to reduce the number of simulations acquired to get a good solution. As the algorithm combines both a surrogate model and parallelism, it is suitable for large-scale problems with expensive function evaluation.

The flowchart of Parallel Stochastic Radial Basis function (p-SRBF) is shown in Fig. 2.1. In each iteration, we generate Candidate points and select P points among a large pool of candidate points for the following expensive function evaluations.Once all P cores finish evaluating expensive functions of the points. We can determine if we find a better point than the current best solution or not. (i.e. find $x^* = \underset{x}{\operatorname{argmin}} F(x)$ for $x \in D_i$ with $length(D_i) = P$ and determine if $x^* < x_{best}$ or $x^* > x_{best}$)

If any x in D_i has better value than x_{best}, we consider iteration i as a successful iteration. But if no improvement in solution set D_i is found (ie. $x^* > x_{best}$) within a certain number of consecutive iterations, the algorithm search will shrink to a

small area (i.e. searching radius σ_i is reduced). Once the number of shrinkage reaches a certain level, p-SRBF restarts from scratch to avoid being trapped in a local minimum.

The modification in algorithm p-SRBF compared to ParLMSRBF is on the pattern of searching radius (i.e. σ_i) as shown in Step (a) in function **Parameters_Update**. Algorithm p-SRBF increases the frequency of reducing the search radius in ParLMSRBF by counting each failed iteration as N failures (failed iteration is defined as the iteration that has no better solution found than the current best one so far) instead of one failure in ParLMSRBF, and N is defined as the number of cores divided by 2. In this way, more information of evaluated points can be used in the search. And also to compensate the less frequent update of response surface as it is updated every P function evaluations instead of every one function evaluation in serial case, we choose $P/2$ as the update for failure iterations (instead of a larger number such as P) in order to allow more iterations to be evaluated before possible triggering a restart which is determined.

function $Flag$ = **Parameters_Update**
(x_i^*, σ_{i-1},$\mathcal{T}_{fail}$,$\mathcal{T}_{success}$,$\mathcal{C}_{fail}$,$\mathcal{C}_{success}$,r_{fail},r_{max},P)
(a) *(Update counter)* **if** $F(x_i^*) < f_{best}$ **then** reset $\mathcal{C}_{success} := \mathcal{C}_{success} + 1$ and $\mathcal{C}_{fail} := 0$;
else reset $\mathcal{C}_{fail} := \mathcal{C}_{fail} + P/2$ and $\mathcal{C}_{success} := 0$;
(b) *(Adjust step size)* **if** $\mathcal{C}_{fail} \geq \mathcal{T}_{fail}$ **then** $\sigma_i = max(\sigma_{i-1}/2, \sigma_{min})$, $\mathcal{C}_{fail} = 0$ and $r_{fail} = r_{fail} + 1$ **else if** $\mathcal{C}_{success} \geq \mathcal{T}_{success}$ **then** $\sigma_i = 2\sigma_{i-1}$, and $\mathcal{C}_{success} = 0$;
(c)*(Check on restart)* **if** $r_{fail} > r_{max}$ **then** Flag = True **else** Flag = False

The pySOT toolbox contains p-SRBF and it is implemented in python. The parallelism uses MPI (Message Passing Interface which is a standardized and portable message-passing system for parallel computing architectures) by a module called MPI4PY, a python MPI package. The parallelism scheme of the algorithm is

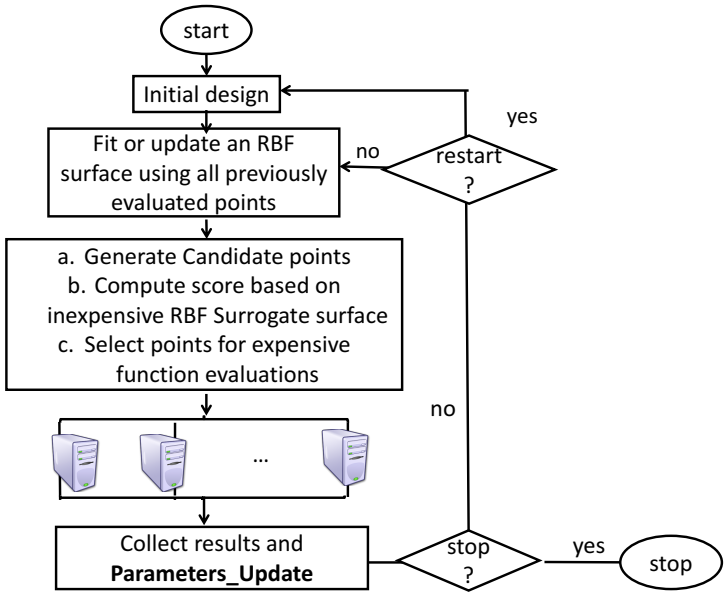


Figure 2.1: Flow chart of p-SRBF

in "master-slave" form. In each iteration, a "master" core distributes P candidate points selected by the RBF surface and distance criteria to P "slave" cores in order to generate real function evaluation values. We use blocking point-to-point communication ($Send()$ and $Recv()$) in MPI to ensure proper synchronization. Then, after the Master core receives simulation results from all "slave" cores, a new RBF response surface can be built based on all results obtained. In our experiment, the termination criterion is the maximum evaluation allowed.

One thing to notice is that when P cores are used, the algorithm uses the P simulation results to update the surrogate surface in each iteration. Also, the best decision vector from all previous evaluations, including the P simulation results in the current iteration is used as the "best solution so far" point from which future decision variables are perturbed. Therefore, the searching pattern is different if a different number of cores used, since (a) the response surface is updated only every P function evaluations, and (b) the way of choosing the next points for evaluation

is based on a different amount of information (i.e. different number of previously evaluated points) in each iteration. Hence, the parallel algorithm can end by giving different answers with same number of evaluations for different values of P.

2.2.2 Alternative Parallel Algorithms

2.2.2.1 Parallel NOMAD

Nonlinear Optimization by Mesh Adaptive Direct Search (NOMAD) is a widely used application to black-box function. As its name implies, NOMAD generate iterations on a tower of underlying meshes on the decision domain and performs an adaptive search on the meshes including controlling the refinement of the meshes. At each iteration, it is composed of two steps, called search step and poll step. Search step is a flexible search on the entire underlying meshes, while poll search is more of a local exploration as it searches only in the vicinity of the current solution.

In our study, we used NOMAD black-box optimization software [1], and applied Parallel NOMAD (p-MADS), the basic parallel version of MADS algorithm in the groundwater problem. The parallelism of NOMAD is based on MPI (message passing interface), and is implemented in C++, but it can take user defined functions in any languages. In addition, we added the option of combining the variable neighborhood search (VNS) algorithm in NOMAD to facilitate escaping from local minima, as VNS keeps moving to a new distant neighborhood of the current solution as long as a better solution is found. NOMAD terminates when its default termination criteria is met, which is when the mesh size has reached NOMAD precision [4]. In order to facilitate comparison with other algorithms, we

added the maximum allowed function evaluations as a second stopping criteria.

2.2.2.2 Asynchronous Parallel Pattern Search

The asynchronous parallel pattern search (APPSAPCK) method solves for nonlinear optimization algorithm and does not require any gradient information. ([35], [25], [45]). In the APPSPACK algorithm, the best point x among the initial design $S_{initial}$ is selected to generate trials points set T based on different step lengths along corresponding search directions. A conveyor is used to distribute the points in set T to different cores, and then gather the evaluated results E in an asynchronous way. If any of the results in E is found to be better than x, this iteration is defined as successful, otherwise it is unsuccessful. In a successful iteration, x will be replaced by the best point found so far, while in an unsuccessful iteration, the search domain is narrowed down by reducing the step length. This search pattern iterates until the termination criterion is met, which is when step length decreases to its minimal length defined. We coupled the C++ objects of APPSPACK with our groundwater model interface in Python, while parallel communication is also based on MPI. Given a limited number of function evaluations, APPSPACK has an early stop if its stopping criterion of length of searching step converge is met, which means APPSPACK is trapped in some locally optimal solution.

2.2.2.3 Parallel Genetic Algorithms

The Genetic Algorithm is a widely known optimization algorithm which is based on the idea of mimicking the search process of natural selection. It solves optimization problems through evolving the best solution from an initial set of random guesses. In the evolution, different strategies can be applied. One strategy is "crossover", a

recombination of the "genes" (elements of the decision variables) so the offsprings can inherit the merits of "genes" from the parents. Offspring can also mutate, as "mutation" is a strategy to add diversity into the population, Then the pairs of parents for the next generation are chosen based on their "fitness" (function evaluation values). As generations increase, the population evolves to the solution. In serial GA, the fitness values are computed one at a time, while for Parallel GA, fitness values are computed simultaneously by employing all the accessible computing cores.

In this work, we employ the Distributed Evolutionary Algorithm in Python (DEAP) module [19] and build the parallel real GA. DEAP is a python toolbox for the Evaluation Strategy combined with MPI. In the toolbox, we chose two points crossover, uniform mutation and tournament selection. Parameters such as crossover probability, mutation rate, and member in tournaments are set to the same as default values in MATLAB genetic algorithm toolbox.

2.3 Study Problems

Our test problems are two U.S. Environmental Protection Agency (EPA) groundwater superfund sites. The first case is at Umatilla Chemical Depot (UCD), Oregon and the second case is at Former Blaine Naval Ammunition Depot (NAD), Nebraska. These two field sites are study problems in a demonstration project. Detailed DoD/ESTCP study report, groundwater flow and transport model and data can be found on project website.(http://www.frtr.gov/estcp) The goals of both two problems (Umatilla and Blaine) are to reduce the contamination residues by the end of management period at minimum cost, with constraints including al-

lowable extraction and injection, treatment plant capacity, as well as mass balance of pumping strategy. In this study, we set the objective to minimize the sum of life-cycle cost of the pump and treat system with an additional penalty cost on constraints violations. Decision variables of the objective function are pumping rates of different pumping wells. Due to limited pumping capacity, decision variables are bounded within a box constraints. Detailed problem formation is provided in Section 2.3.1.2 and 2.3.2.2.

2.3.1 Case 1: Umatilla Chemical Depot (UCD)

2.3.1.1 Site Background

Umatilla Chemical Depot (UCD) consisted of a 19,728 acres military reservation established in 1941 as an ordnance depot. During the operation as onsite explosives washout plant from the 1950s, the depot disposed wash water from the plant into two unlined lagoons, where wash water with its containing explosives was infiltrated into the soil and contaminated groundwater system. The hydrogeology for Umatilla depot site consists of a confined alluvial aquifer overlying silt and weathered basalt in convertible state between confined and unconfined aquifer. The hydraulic conductivity in the aquifer is highly heterogeneous varying from 1ft/day to 5000 ft/day. The simulation of the groundwater flow and transport is based on USGS MODFLOW2005 ([31]) and MT3DMS 5.3 ([79]) models. Two pollutants used as indicating parameters are Royal Demolition Explosive (RDX) and 2,4,6-Trinitrotoluene (TNT) due to their high concentration relative to other contaminations.

2.3.1.2 Model formulation

The formulation of remediation problem on Umatilla site is based on Formulation 1 in [47] and [66]. The objective is to minimize the operation cost during the remediation process of 4 years, with goal of reducing contamination residues to certain levels by the end of management period. Other constraints include allowable extraction and injection, treatment plant capacity, as well as mass balance of pumping strategy. We used a pumping system of fixed eight pumping wells and two recharge basins. Ideally, concentration of TNT after 4 years should be less than $2.1\mu g/l = C_{TNT}$ and concentration of RDX after 4 years should be less than $2.8\mu g/l = C_{RDX}$, and the exceedance of these two cleanup levels is given as penalty cost in the objective function.

The objective function is shown as below,

$$\underset{Q}{Min}(V_{CE}(Q) + V_{CG}(Q) + PenaltyCost(Q, C)) \tag{2.1}$$

Subject to the constraint:

$$Q_i^{min} \leq Q_i \leq Q_i^{max} \qquad \forall i \in [1, 2, ...10] \tag{2.2}$$

where, $Q = (Q_1, Q_2, ..., Q_{10})$ is a vector of 10 decision variables specifying pumping or recharging rates at the 10 wells in the system. The Q_i are subjected to constraints that the extraction/injection pumping capacity of each pumping well i cannot exceed 400 gpm $(Q_1, Q_2..Q_8)$, while the pumping capacity of each recharging well i cannot exceed 800 gpm (Q_9, Q_{10}). V_{CE} is Net Present Value (NPV) of variable electrical cost of operating wells. V_{CG} presents the NPV of mass removal using GAC unit. $PenaltyCost$ in Equation (2.1) is associated with constraint violations on unsatisfied contaminant clean up levels at the end of overall management period and exceedance of total pumping capacity. In addition, the total amount

of pumping and the total amount of recharge should be balanced at the end of management period, and a penalty will be assigned to the discrepancy.

$$PenaltyCost = w_1 * Viol_{C_{TNT}} + w_2 * Viol_{C_{RDX}} + w_3 * Viol_{TotalPumping} + w_4 * Viol_{MassBalance} \quad (2.3)$$

where C_{TNT} and C_{RDX} are the maximum permissible concentration of those two contaminants. $Viol_{C_{TNT}} = \max(C_{TNT} - 2.1, 0)^2$, $Viol_{C_{RDX}} = \max(C_{RDX} - 2.8, 0)^2$, $Viol_{TotalPumping} = \max(\sum_{i \in Pump} Q_i - Q_{sum}, 0)^2$, and $Viol_{MassBalance} = (\sum_{i \in pump} Q_i - \sum_{i \in recharges} Q_i)^2$

2.3.2 Case2: Blaine Naval Ammunition Depot (NAD)

2.3.2.1 Site Background

Blaine Naval Ammunition Depot (NAD) has 48,800 acres located east of Hastings, Nebraska. It was built in the early 1940s as an active ammunition facility and maintained high production during World War II and the Korean Conflict. Later during its subsequent decommissioning process (1958-1967), waste water were discharged into surface impoundments and natural drainage areas of the facility, which consequently contaminated the groundwater. The hydrological condition at Blaine Ammunition site consists of an unconfined aquifer consists of sand, gravel and clay lying above a thin upper-confined layer and a thick semi-confined aquifer. The groundwater flow and transport Model is also simulated based on USGS MODFLOW2005 [31] and MT3DMS 5.3 [79]. Contaminations of concern are VOCs and explosives, and in this study, Trichloroethylene (TCE) and Trinitrotoluene (TNT) are chosen as two parameter indicators.

2.3.2.2 Model formulation

The Blaine remediation problem has a similar objective goal to the Umatilla problem, which is to minimize the cost for entire project duration and the penalty cost associated with constraint violations. It is based on Formulation 1 of Blaine problem in [47]. In our study, we use the modeling period as 30 years containing 6 management periods, each of 5 years. The formulation contains 15 pumping wells as decision variables with each of them kept constant throughout the entire six management periods. The simulation is more complex than Umatilla site due to its longer management periods and more pumping rates involved. The objective function is to minimize the management cost including fixed cost of facility installation once per each management period and maintenance and operation cost for the whole project duration. There is a cleanup level constraint for each of the indicator contaminants at the end of the project, i.e. concentration of TNT after 30 years should be less than $2.8\mu g/l$ and concentration of TCE after 30 years should be less than $5\mu g/l$.

Its objective function is shown as below,

$$\underset{Q}{Min}(C_{CE} + C_{CT} + C_{CD} + F_{CM} + F_{CS} + V_{CE} + PenaltyCost)$$

Subject to the constraint:

$$Q_i^{min} \leq Q_i \leq Q_i^{max}$$

Where, $Q = (Q_1, Q_2, ..., Q_{15})$ is a vector of 15 decision variables presenting pumping rates all subjected to constraints that pumping capacity of each pumping well cannot exceed $350gpm$. C_{CX} is capital cost of installing treatment unit, such as extraction well (C_{CE}), treatment facility (C_{CT}), discharge pipe (C_{CD}). F_{CX} is fixed cost which consists of fixed cost for management (F_{CM}) and fixed cost of

sampling (F_{CS}). V_{CE} is variable electric cost of operation depends on pumping rate Q. *PenaltyCost* is associated with constraint on meeting clean-up level based on two criteria; one constraint is preventing polluting uncontaminated areas above the cleanup level during the remediation process, and the other constaint is to satisfy contaminant clean up levels at the end of overall management period for both two chemicals.

$$PenaltyCost = w_1 * Viol_{C_{TCE}} + w_2 * Viol_{C_{TNT}} + w_3 * I_{TCE} + w_4 * I_{TNT} \quad (2.4)$$

where $Viol_{C_{TCE}} = \max(C_{TCE} - 5, 0)^2$, $Viol_{C_{TNT}} = \max(C_{TNT} - 2.8, 0)^2$, I_X is the number of exceedance of its cleanup level for contamination X in the spatial modelling domain among all stress period.

2.4 Experimental Set up

2.4.1 Supercomputer resources set up

The computational experiments in this study were performed on Yellowstone Supercomputer, a high-performance cluster at National Climate Atomespheric Research Center (NCAR). On this platform, each trial run of the simulation and optimization was sent as a scheduled job through platform Load Sharing Facility (LSF) in a batch script file, which defines the number of total tasks (N_{tt}) in the job and number of taskes per nodes (N_{tpn}). In this context, a task is the smallest unit of work that can be handled by a computing core, however a job is a logical ensemble of tasks [14]. As each node processes multiple tasks, we are able to set up the number of nodes needed as N_{tt}/N_{tpn} in job submission and the number of cores used is N_{tt}. In addition, since simulation on MODFLOW writes out a

flow-transport link file required to be read by MT3D, we generate separate folders for each core used, to avoid conflicts of accessing the same file when using multiple cores simultaneously. As a result, the storage requirement can be large, especially for a large model as Blaine, which can require approximately 50GB when using 32 cores simultaneously.

2.4.2 Trials set up

The randomness brought by the stochastic nature of the algorithm requires a rational assessment based on multiple trials. Therefore, we conduct a minimal 10 trials for each experimental setting. Each experimental setting varies by either having a difficult number of core or different algorithm. All settings run for several trials and have the same set of multiple initial designs to start with. Scalability analysis of p-SRBF is formed on the exact same initial design using different number of core. For different algorithm comparison, cases may vary. p-SRBF, Parallel NOMAD and its variation with VNS take the same initial design i.e. a set of initial starting points $S_{initial}$. APPSPACK starts with the best Points found in $S_{initial}$. And the initial population of Parallel GA uses a random uniform design for population points with the worst individual replaced by the best initial point from $S_{initial}$.

2.5 Results and Discussion

2.5.1 Result Quality

The optimal solutions found by p-SRBF are compared with the previous studies conducted by MGO group [80]. On Umatilla problem, we compared our solutions for optimal pumping rates to those obtained by MGO group on the same problem. We did not try to optimize the locations of the pumping wells and instead used all the possible well locations in MGO group. Our p-SRBF method aims at reducing the variable cost only. The summary of comparison of the optimal solutions is shown in Table 2.1 with items listed on the Umatilla problem. The two optimal solutions turn out to have the same configuration of pumping wells, and p-SRBF finds an optimal pumping strategy with lower variable costs. In addition, the optimal solution from p-SRBF and from MGO group produce a concentration level satisfying the requirement for both RDX and TNT, which are respectively $2.09\mu g/l$ and $2.75\mu g/l$ at the end of 4 years.

For the Blaine problem, we use the total cost including both fixed costs and variable costs as our objective function, which aligned with what the MGO group used. However, our study is focused on a steady pumping strategy which has a constant pumping rate along all management periods, and optimization in the MGO group is aimed at a dynamic pumping strategy. By evaluating the two optimal solutions, p-SRBF achieves a lower cost than MGO. The detailed comparison in shown in Table 2.2.

Table 2.1: Optimal pumping strategy by p-SRBF as compared with MGO group on Umatilla Site ((L,R,C) represents (Layer, Row, Column))

Name	Location (L,R,C)	Pumping/Injection Rate(GPM)	
		MGO	p-SRBF
EW-1	(1,60,65)	-307.5	-339.3
EW-2	(1,83,84)	0	0
EW-3	(1,53,59)	-219.5	-396.0
EW-4	(1,85,86)	0	0
NEW-1	(1,48,59)	-360	-392.0
NEW-2	(1,48,55)	-283	-41.0
IF-1	*	0	0
IF-2	*	380	420
IF-3	*	790	747
IF-L	*	0	0
Electricity variable cost		$48,394	$43,491
GAC unit variable cost		$11,700	$5,726

2.5.2 Graphical Parallelism Analysis

A graphical depiction of parallelism performance is presented in the Progress Graph and the Wall-clock-time Progress Graph, as shown in Fig. 2.2 to Fig. 2.5. Fig. 2.2 and Fig. 2.3 are Progress Graph and Wall-clock-time Progress Graph for Umatilla problem and Fig. 2.4 and Fig. 2.5 are for Blaine problem.

The Progress graph plots the average best objective function value found across different trials (i.e $\overline{R_{P,j}}$ for P as core count, j as trial number index) against the number of objective function evaluations and the value $\overline{R_{P,j}}$ is updated every P function evaluation at each j^{th} trail. This can be seen in Fig. 2.2 that for 128 cores, the best solution is visibly as step function with the step lasting 128 or more evaluations, whereas the best solution for the serial case (1 core) appears as a smooth line. The Wall-clock time Progress Graph describes speedup by plotting the best average function value among trials against the wall clock time at which the value is reached. Since most of computation time is for simulation and simula-

Table 2.2: Optimal pumping strategy by p-SRBF as compared with MGO group on Blaine Site ((L,R,C) represents (Layer, Row, Column))

Well ID	Location (L,R,C)	Pumping/Injection Rate(GPM)						
		MGO						SRBF
		P1	P2	P3	P4	P5	P6	P1-P6
1	(3,27,59)	-350	-15	-50	-45	-350		
2	(3,35,78)	-290	-170	-180	-305	-350		-178.4
	(4,35,78)	-290	-170	-180	-305			-178.4
	(5,35,78)	-290	-170	-180	-305			-178.4
3	(3,52,120)	-295	-240	-275	-240			-115.4
4	(3,47,112)	-120	-330	-310	-155			-180.7
	(4,47,112)	-120	-330	-310	-155			-180.7
5	(3,27,38)	-66	-170	-100	-100	-50	-50	-9.5
6	(3,39,36)	-147	-79	-231	-215			-345.2
7	(3,28,61)		-286	-225	-190	-275	-330	
	(4,28,61)		-286	-225	-190	-275	-330	
8	(3,30,65)		-254	-110	-125	-350	-325	-68.3
	(4,30,65)		-254	-110	-125	-350	-325	-68.3
9	(3,57,109)		-350	-350	-185			-239.7
10	(3,31,70)			-260	-215	-350		-157.8
	(4,31,70)			-260	-215	-350		-157.8
11	(3,32,62)				-315	-200	-350	-324.5
	(4,32,62)				-315	-200	-350	-324.5
12	(3,26,55)					-300	-345	-234.1
	(4,26,55)					-300	-345	-234.1
13	(4,32,75)					-200		
	(5,32,75)					-200		
14	(3,27,32)						-330	-90.7
	(4,27,32)						-330	-90.7
15	(3,27,30)						-170	-72.7
	(4,27,30)						-170	-72.7
total cost		50030						48631
TNT concentration		2.78						2.49
TCE concentration		4.99						5.01

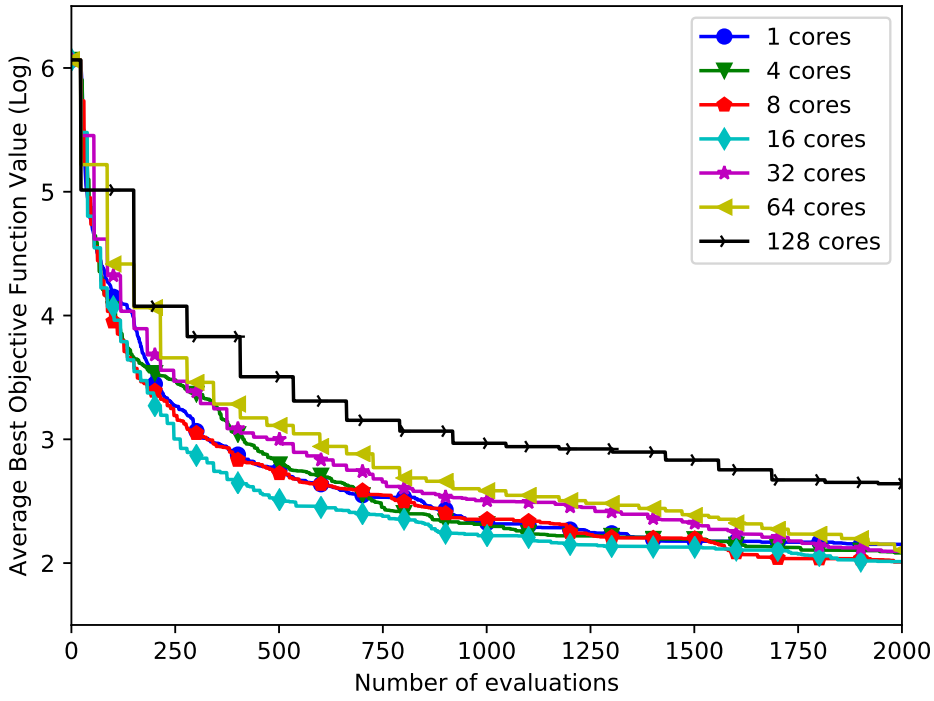


Figure 2.2: Progress Graph of average value of function evaluation among 30 trials vs. number of function evaluations on Umatilla Site

tion computational time (t_s) does not vary much among the cases having different inputs, the number of function evaluations can also be viewed as an approximation of the CPU time used in the system (i.e. $CPUtime \approx t_s * P$).

For the Umatilla problem, at the end of 2000 function evaluations, the parallel processing computation can attain a similar accuracy level to its serial version. In Fig. 2.2, core counts of 4, 8, 16, 32 reach slightly better (or smaller) averaged function values than the serial, while the case using 64 cores catches up with serial cores with 2000 function evaluations as well. Noted that obtaining the same function value at the same number of evaluation for parallel and serial case indicates that this core count reaches close to 100% efficiency at this accuracy level. Fig. 2.2 also shows that the case with 16 cores outperforms all other cases in terms of best solution versus number of evaluations as it stays as the lowest curve during

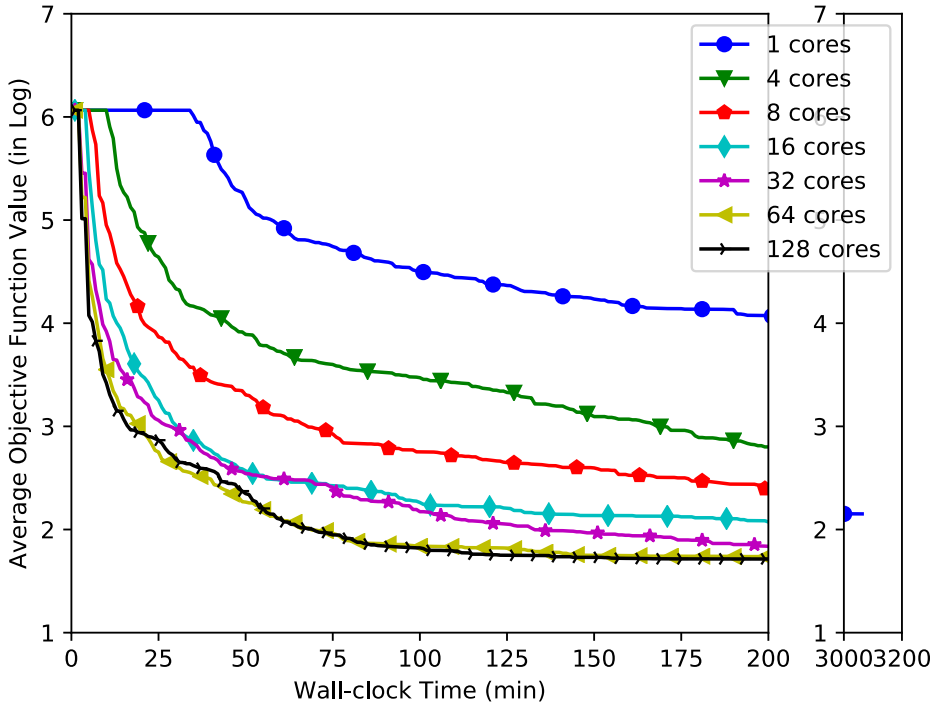


Figure 2.3: Wall-clock Progress Graph of average value of function evaluation among 30 trials vs. wall-clock time on Umatilla Site with small subplot of serial result as reference level at 2000 evaluations

the function evaluations. This is an outstanding performance of the parallel algorithm compared to its serial version. (Typically the serial version performs better than the parallel versions in terms of number of evaluation required.) At the same amount of wall clock time shown by Fig. 2.3, the trend shows that more cores used could reduce the time required to obtain a better solution. However, the case of using 128 cores performs as same as core count 64, which shows that increasing the number of cores beyond some limit (in this case 64) does not necessarily guarantee an increase in efficiency. This can be due to many factors including the infrequent updates in the surrogate to possibly increased memory requirements.

The Blaine Aquifer problem (30 min per objective evaluation) is more computationally expensive. Consequently, we present the result with (a limited) 400

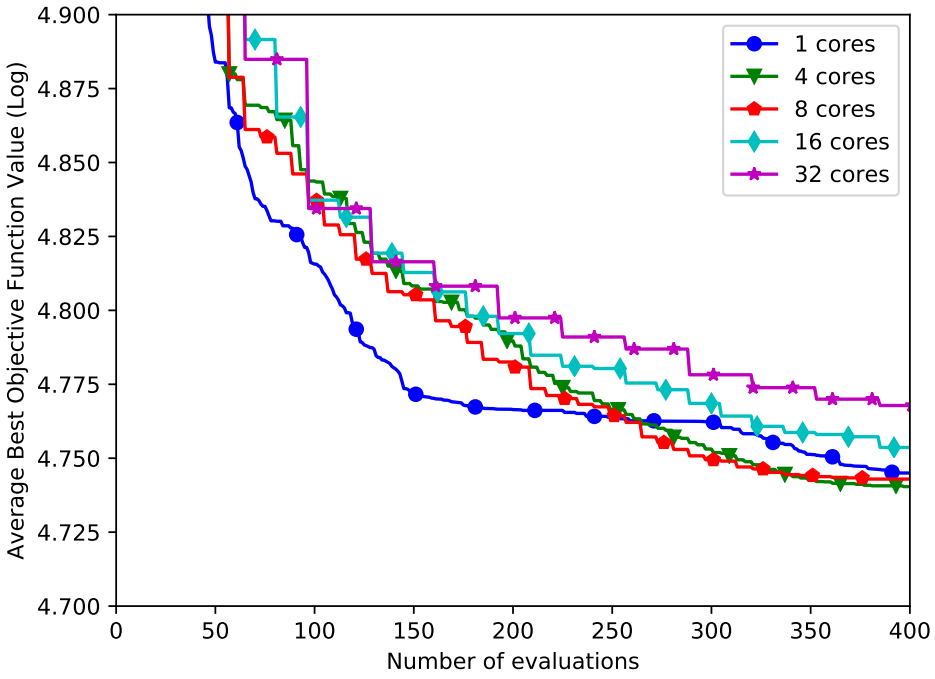


Figure 2.4: Progress Graph of average value of function evaluation among 10 trials vs. number of function evaluations on Blaine Site problem

function evaluations. This requires 8 days to complete when using the serial algorithm. Even though the serial case of Blaineis more efficient (in terms of evaluations versus best cost) than the parallel version in the progress graph at the first 200 function evaluations, its lengthy execution time makes it far less competitive compared to its parallel version, as shown in Fig. 2.5. Moreover, at the end of 400 function evaluations, the results are close for all cases, which shows a good performance for parallel processing up to 32 cores. A detailed statistical comparison is presented in Section 2.5.4. When comparing among different core counts, 4 and 8 are very efficient as reaching a result level below serial, and core count 32 outperforms all the settings, when comparing at the same wall-clock time. For the case using 16 cores, it performs efficiently at the beginning, but levels out after 800 minutes, which illustrates an unpredictable performance aspect to the parallel processing.

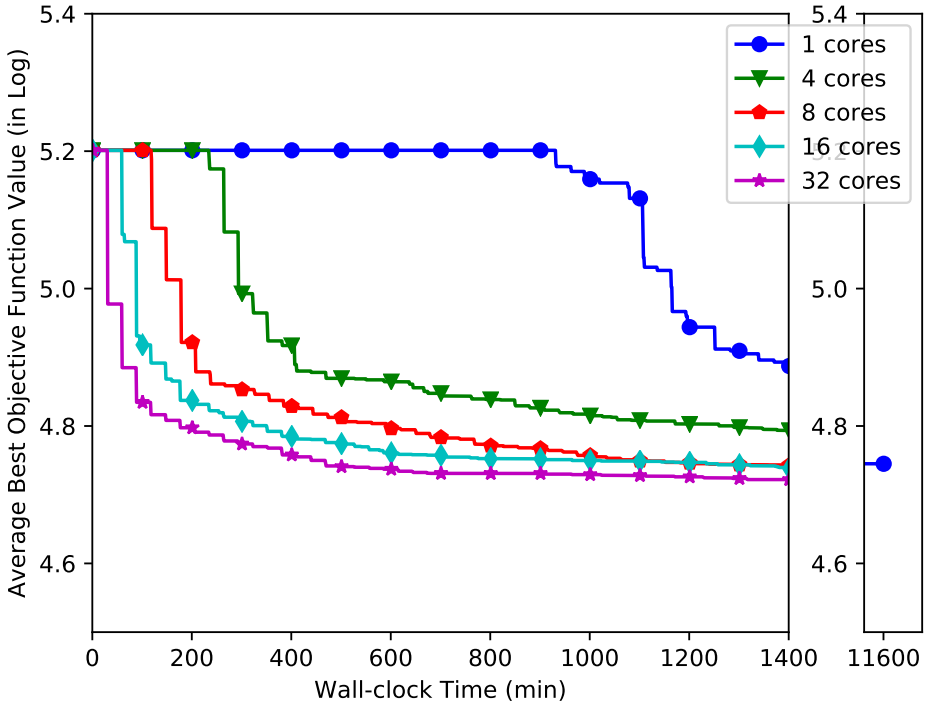


Figure 2.5: Wall-clock Progress Graph of average value of function evaluation among 10 trials vs. wall-clock time on Blaine Site problem with small subplot of serial result as reference level at 400 evaluations

2.5.3 Quantitative Parallelism measure

To quantify the analysis of the parallel efficiency, speedup and efficiency are used as two common measures in parallel computations. In our study, we first define different threshold levels (L) as references for evaluation. Due to the stochastic nature of the algorithm, we analyze the effectiveness of parallelism in terms of its averaged solution across the trials, and compute the Wall-clock time for the averaged solution to reach to the level (L_i). Speedup is then defined as the ratio of wall-clock time required by serial algorithm (T_{serial}) to the wall-clock time required by its parallel version using P cores ($T_{parallel}(p)$) to reach to the same accuracy level (L_i). Efficiency is the ratio of speedup to the number of core counts P. Equations for speed up and efficiency are presented in Equations 2.5 and 2.6.

Table 2.3: Speed up (SP) and Efficiency (E) of p-SRBF on Umatilla problem

Cores	Level 1		Level 2		Level 3	
	SP	E	SP	E	SP	E
4	4.26	106%	4.20	105%	4.58	114%
8	7.43	92%	7.69	96%	8.86	110%
16	19.2	120%	20.81	130%	21.29	133%
32	20.64	64%	24.6	76%	28.45	88%
64	38.04	59%	47.47	74%	51.91	81%
128	35.93	28%	48.32	38%	51.91	40%

$$Speedup(p) = \frac{T_{serial}(\overline{R_{serial,j}} \leq L)}{T_p(\overline{R_{p,j}} \leq L)} \tag{2.5}$$

Efficiency is then defined as

$$Efficiency(p) = \frac{Speedup(p)}{p} = \frac{T_{serial}(\overline{R_{serial,j}} \leq L)}{T_p(\overline{R_{p,j}} \leq L) * p} \tag{2.6}$$

A summary of speed up and efficiency for different core counts is provided in Table 2.3 for Umatilla problem and Table 2.4 for Blaine problem. The baseline of reference levels is chosen as the best objective value obtained in serial algorithm averaged across trials ($\overline{R_{serial,j}}$ for $j \in (1, 2, ..N)$). For Umatilla problem, three accuracy levels are $L1 = 5\%$ above $\overline{R_{serial,j}}$, $L2 = 1\%$ above $\overline{R_{serial,j}}$, $L3 = \overline{R_{serial,j}}$. For Blaine problem, three accuracy levels are $L1 = 0.5\%$ above $\overline{R_{serial,j}}$, $L2 = 0.3\%$ above $\overline{R_{serial,j}}$, $L3 = 0.1\%$ above $\overline{R_{serial,j}}$. The order of L1 to L3 shows an increasing difficulty of attainability.

For easily attainable threshold value, it represents the performance at the early stage of the run, and for small threshold value which is hard to be obtained, its associated speed-up and efficiency measured at the end of run. From Table 2.3 and Table 2.4, we can see that for both two problems, p-SRBF is more efficient in terms of finding a more accurate optimal solution. In the case when we use a small number of cores, we are able to reach efficiency of above 100%, especially for

Table 2.4: Speed up (SP) and Efficiency (E) of p-SRBF on Blaine problem

Cores	Level 1		Level 2		Level 3	
	SP	E	SP	E	SP	E
4	2.77	69%	4.45	111%	4.62	116%
8	5.87	73%	9.36	117%	9.67	121%
16	9.04	57%	14.86	93%	9.74	61%
32	14.30	45%	23.92	75%	22.52	70%

case with 16 cores which attains 133% efficiency on Umatilla and 8 cores attains 120% on Blaine. This is super-linear performance (i.e. speedup above 100%), which means that the speedup with P cores is more than P times faster than the same optimization performed on single core to reach the same accuracy level. Since the next evaluation point selected depends on P and the surrogate is only updated every P evaluations. A different number of cores employed changes the algorithm, so the search pattern is no longer the same as in the serial case. As to cases of large core counts employed, the efficiency for 32 cores are 45% to 70% to reach the difficult level as we can observed from the Table 2.3 and 60% from Table 2.4. The performance shows that even though the CPU time required to reach to the same result as in the serial case has increased with certain large number of cores employed, we are still able to reduce the execution time by a very significant amount.

The superlinear speedup performance of the algorithms is related to change of the searching pattern in the algorithm. We analysis the performance of the restart strategy to help us understand the modification in algorithm. Table 2.5 summarizes the number of restarts when using different number of cores. We can see that we have reduced number of restarts when having more cores used, but the averaged reduction in restarts is not inversely proportion to the increase of number of cores, which in a way indicating the change of searching pattern. And we obtain

Table 2.5: Summary of averaged number of restarts ($\mu_{restart}$) and standard deviation of number of restarts ($\sigma_{restart}$) occurred within 2000 function evaluations in Umatilla problem among 30 trials, and $\mu_{restart}$ and $\sigma_{restart}$ occurred within 400 function evaluations in Blaine problem.

Problem	cores	$\mu_{restart}$	$\sigma_{restart}$
Umatilla	1	13.6	0.8
	4	7.1	0.4
	8	5.7	0.5
	16	4.2	0.4
	32	1.9	0.3
	64	0.9	0.2
	128	0	0
Blaine	1	1.9	0.3
	4	0.6	0.5
	8	0.9	0.3
	16	0.3	0.5
	32	0	0

a small standard deviation of number of restarts among 30 trials for each case with a certain number of cores used. So the restart strategy performs consistently for both problems. Therefore the way of reducing searching radius as in function **Parameters_Update** helps the p-SRBF to thoroughly explore the area from the initial design when using large pool of cores, and still enables the restart to avoid the solution being trapped in some local optimum.

2.5.4 Stochastic Performance analysis

Section 2.5.3 presents averaged efficiency of p-SRBF based on the averaged solution across multiple trials. As the efficiency varies from trial to trial due to stochastic search in the algorithm, statistical analysis on the performance over trials is conducted to evaluate the robustness of the performance.

Table 2.6: P-value in Wilcoxon Rank-Sum Test on various core counts to test whether results using P number of cores are sigificantly different than the results using single core at N function evaluations (i.e. 1000 or 2000) among 30 trials on Umatilla problem

Cores	1000 evaluations	2000 evaluations
4	0.8592	0.8245
8	0.8360	0.3147
16	0.5444	0.4965
32	0.6152	0.7338
64	0.8016	0.3593
128	0.0029*	0.0646

* means the results are significantly different from using single core at $\alpha = 5\%$

The statistical comparison analysis is employed on the results obtained at the same number of function evaluations. Table 2.6 presents the summary of p-value in two tails Wilcoxon Rank-Sum Test comparing the objective value between the parallel processing and the serial case at 1000 evaluations and 2000 evaluations. Comparison of results obtained at same evaluations using different number of cores does not consider the benefits of reduction in wall clock time with multiple cores. At 5% significance level and at both 1000 and 2000 function evaluations, there is no significant difference between results obtained by serial and the results from cases using 4, 8, 16, 32, 64 cores. And even though the 128 cores results are significantly different from serial result at 1000 evaluations, they are not significantly different at 2000 function evaluations according to statistical test. Similar improvement in performance can be found on Blaine problem using 4, 8 and 16 cores in Table 2.7, where the results across trials become similar to its serial algorithm at 400 function evaluations. However, results by using 32 cores are significantly different from the serial results at 5% level.

Fig. 2.6 shows the box plot of objective function values for different core counts. Subplot (a) and (b) present results on Umatilla problem at 100 minutes

Table 2.7: P-value in Wilcoxon Rank-Sum Test on various core counts to test whether results using P number of cores are sigificantly different than the results using single core at N function evaluations (i.e. 200 or 400) among 10 trials on Blaine problem

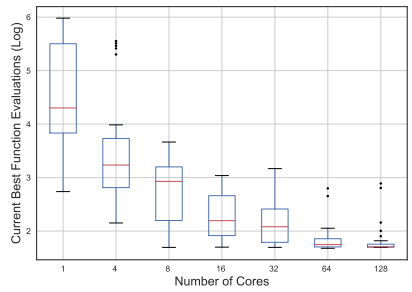
Cores	200 evaluations	400 evaluations
4	0.0588	0.6501
8	0.1509	0.6501
16	0.0696	0.5967
32	0.0494*	0.0233*

* means the results are significantly different from using single core at $\alpha = 5\%$

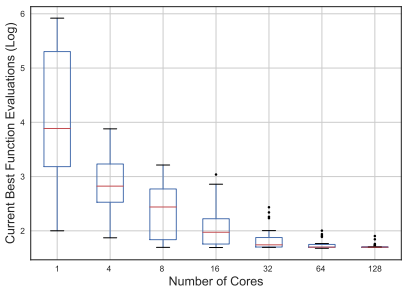
and 200 minutes. As shown in Fig. 2.6, the more cores we use, the smaller median we obtain and the range becomes narrower both at 100 minutes and 200 minutes. The case using 128 cores has the lowest median and narrowest range among all. For both Umatilla and Blaine problem, using a large number of cores shows a consistently effective performance at both the early stage and at the end. Meanwhile its reliability increases with the number of iterations.

2.5.5 Comparison of Four Algorithms

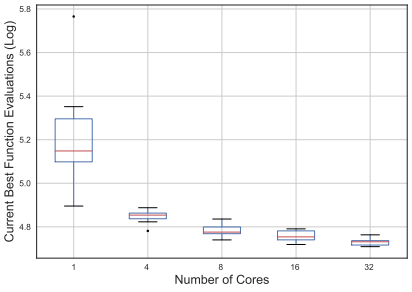
The relative performance of p-SRBF compared with three alternative parallel optimization algorithms is presented in the time analysis plot (Fig. 2.7 and Fig. 2.8). The averaged performance across trials is compared. We conduct the execution for fixed amount of time with the same number of cores for different algorithms. And what we can observe from both Fig. 2.7 and Fig. 2.8 is that p-SRBF shows a dominant performance among all the algorithms, in both Umatillla and Blaine problem. APPSPACK, as a more local optimization algorithm stops early as it reaches its convergence level in Umatilla problem, but it is the second best algorithm performed on these two test problems. Parallel NOMAD and its more



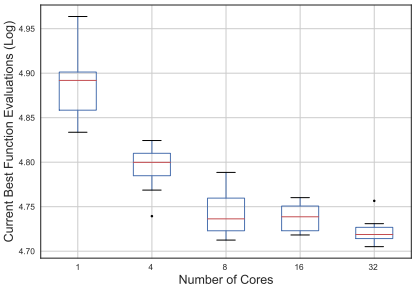
(a) Umatilla results among 30 trials at 100 min



(b) Umatilla results among 30 trials at 200 min



(c) Blaine results among 10 trials at 700 min



(d) Blaine results among 10 trials at 1400 min

Figure 2.6: Box-plot of stochastic performance at the same wall clock time, vertical axis is the objective function value and the red bar is the median value among trials

global version with Variant Neighborhood Search (VNS) have close performance. However, NOMAD with VNS on Umatilla problem reaches better result than NOMAD itself. We deduce the reason is that the optimization on objective function of Blaine is more of a local problem. As to Parallel Genetic Algorithm, it performs the worst on Umatilla and has close performance with Parallel NOMAD on Blaine.

The box plot test the consistency in the behavior of all four algorithms over trials (Fig. 2.9 and Fig. 2.10). In both Umatilla and Blaine problem, p-SRBF shows superiority over all algorithms as it has a smallest quantile bound (i.e. the size of the box) especially and lowest median at the later execution (ie. subplot (b) compared to subplot (a)) for both Umatilla and Blaine problems.

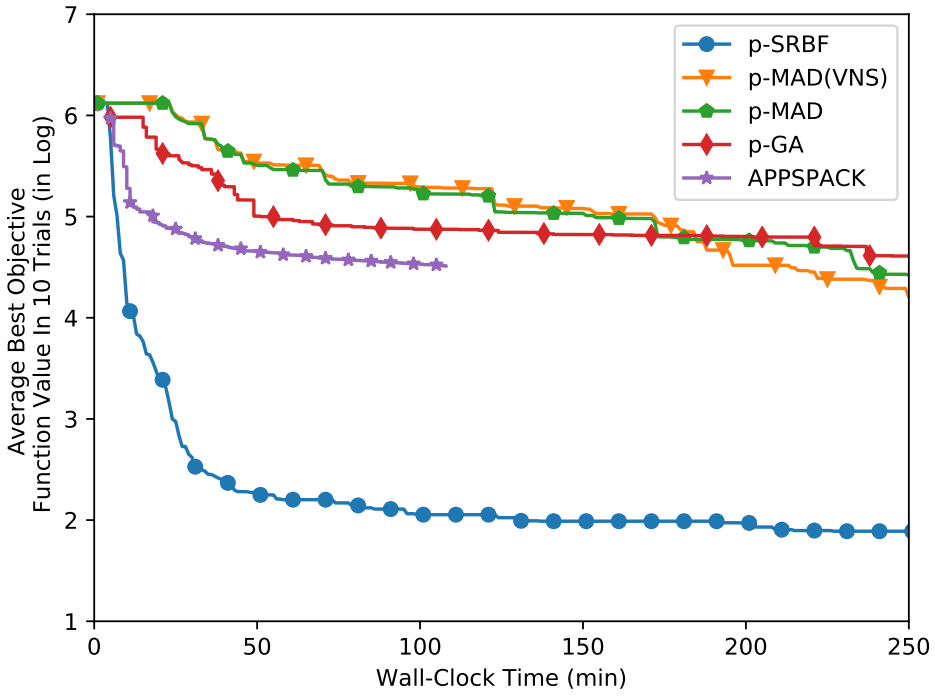


Figure 2.7: Time Analysis Graph of results using different algorithms on Umatilla problem using 16 cores

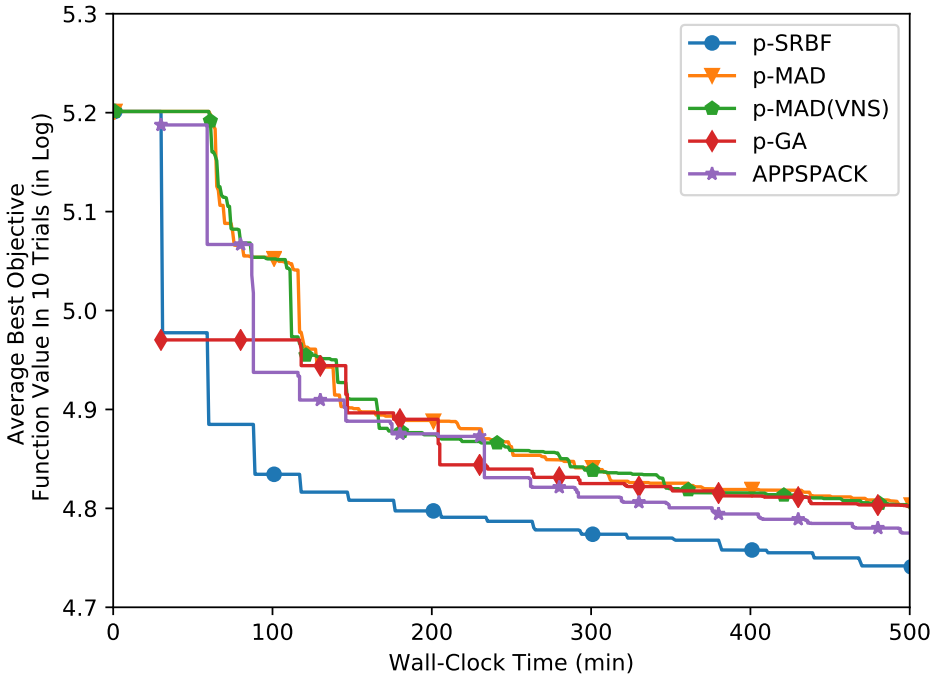
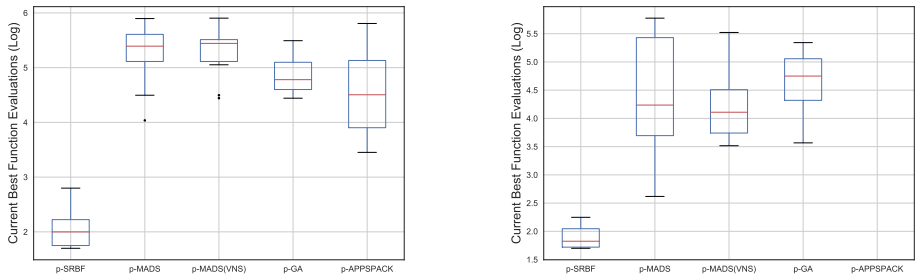
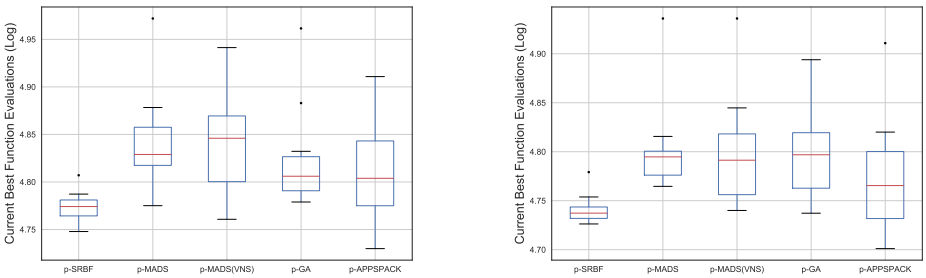


Figure 2.8: Time Analysis Graph of results using different algorithms on Blaine problem using 32 cores



(a) Comparison at 100min (b) Comparison at 250min

Figure 2.9: Box plots of results on Umatilla problem with 10 trials of all algorithms using 16 cores in terms of the same wall-clock time



(a) Comparison at 300min (b) Comparison at 500min

Figure 2.10: Box plots of results on Blaine problem with 10 trials of all algorithms using 32 cores in terms of the same wall-clock time

2.6 Conclusion

This study shows the effectiveness and robustness of the p-SRBF on optimization for computationally demanding groundwater problems. In both case problems, we are able to attain super-linear speedup with less than 8 cores. With a large pool of cores available, we reduce the wall-clock time required to achieve the same accuracy level as its serial version by a large factor. In addition, using more cores can reduce the uncertainty of obtaining a good optimization result. However, more cores does not necessarily guarantee a reduction of computation time, and

computational memory limitation can be an issue. As the search pattern changes due to using different core counts, it is hard to predict the number of cores required for reaching an optimal efficiency. From the computational perspective, p-SRBF can find a better optimal solution with limited budget of computational time. p-SRBF outperforms all the other Parallel derivative-free optimization algorithms in our comparison including the genetic algorithm and mesh adaptive pattern search methods as well as the asynchronous parallel pattern search strategy. The efficient performance of the p-SRBF is consistent across different trials in both the Umatilla and Blane problems, which demonstrates a promising usage in application to other computational expensive simulation models.

CHAPTER 3

OPTIMIZATION OF GROUNDWATER WITHDRAWALS MANAGEMENT PLAN IN PUMPING WELL NETWORK WITH LAND SUBSIDENCE CONSTRAINT USING NEW GLOBAL PARALLEL OPTIMIZATION ALGORITHM

3.1 Introduction

Overpumping groundwater due to rapid development of urbanization and industrialization is an increasing problem around the world. The resulting difficulties include land subsidence, earth fissures, sea water intrusion and groundwater storage depletion [63]. Land subsidence has captured growing attention as it affects many important cities and threatens large populations [70, 52, 6, 38] undesirable consequences brought by land subsidence include increasing localized flooding and erosion, sinkholes, infrastructure damage, and altered conveyance and drainage pathways [53]. In order to alleviate the extent of subsidence, two obvious methods are to reduce groundwater withdrawals or to import artificial recharge [20]. However, in many areas, importing recharge is not possible and reducing total groundwater extraction causes hardship. An alternative is to redistribute water extraction based on spatial distribution of pumping design in order to both obtain a substantial water supply and to reduce induced subsidence in critical area.

Simulation-Optimization (S/O) strategy is an efficient tool for deriving optimal design, however, few studies using S/O framework were focused on designing a pumping plan to alleviate land subsidence problem. All approaches assume subsidence is represented by a linear model [63]. Among them, Larson et al. (2001) [46] applied linear programming with the response matrix technique on a numeri-

cal model of groundwater flow and land subsidence in Los Banos-Kettleman City area of San Joaquin Valley, CA. However, by virtue of the limitation of linear optimization method, the incorporated land subsidence model is constrained in elastic compaction, which means that subsidence is recoverable if the water level rises, but the unrecoverable and inelastic compaction is unable to be considered in their study. Phillips et al. (2003) [53] developed a ground-water-flow model for Lancaster, Antelope Valley, California by using a linear programming optimization to maximize lowest value of head as a consideration of land subsidence. In other words, they did not model subsidence directly but instead, control hydraulic head in order to maintain induced land subsidence. Chang et al. (2007) [8] built a stochastic groundwater management model and approximated the non-smooth optimization into a mixed integer linear programming which includes binary variables to ensure the condition that land subsidence only occurs when drawdown is increasing. The result presented is based on a hypothetical test problem but not a real-world application. The hypothetical problem is a small problem as it only contains five control wells in a aquifer domain discretized into around 100 model cells. In contrast, our test problem is a field application which has around 700 wells in area with more than 60,000 model cells covering an area of 6,500 km^2.

In this study, we adopt MODFLOW [30] and its extension SUB model [34] to simulate the real world groundwater flow and land subsidence caused by groundwater extraction. In the SUB model, the compaction of aquifer is a piecewise linear function of compaction stress. The shape of the function depends on existence of either elastic or inelastic compaction in each location. The vertex of the piecewise function which represents preconsolidation stress (i.e. the largest consolidation stress that has been reached in history) is different at each location. The model is generally non linear and non convex because it is a combination of pumping

at various pumping wells, the integration of the piecewise linear relation between compaction and stress, and the governing equation related to hydraulic head in MODFLOW. Therefore a non-linear global optimizer is more appropriate for this problem.

In addition, the computational budget for the simulation can be large because this integrated large-scale, real-world model is complex with a heterogeneous aquifer, large number of controlling wells, and long-term management cycle. In this study, in addition to using a popular evolutionary algorithm i.e. parallel Genetic Algorithm to solve this real world problem, we also implement parallel version of DYCORS (DYnamic COordinate search using Response Surface models) [60], which is an efficient algorithm suitable for High-dimensional, Expensive, and Black-box problem. DYCORS is a surrogate-based optimization, which is an extension of StochasticRBF [57] and [59] with an additional dynamic coordinate search strategy.

Based on these optimization technology, we obtain groundwater management plans on a $6500km^2$ site that has subsidence risk reduced. The main contribution of this work is to study on using surrogate-based parallel optimization algorithms to solve problems involving land subsidence controlling models which contain a large number of wells.

In addition, we implement a new algorithm DYSOC, which is developed as combination of previous algorithm DYCORS [60], parallel LMSRB [59] using surrogates for expensive constraints. Regis (2011) [56] introduces ConstrLMSRBF algorithm which also contains this expensive constraint handling strategy which integrates response surface for constraint function. Compared with ConstrLMSRBF, DYSOC has a different dimension perturbation strategy and the objective func-

tion to optimize are defined differently with penalty function included. Detailed explanations are in Section 3.5.3. In this study, we show that DYSOC can solve for problems with expensive constraint efficiently based on analysis of efficiency of DYSOC on different formulations which aims at satisfying various extraction requirements. A comparison with the results in parallel GA is also made. In all, our study cases show that parallel DYSOC outperforms both parallel DYCORS and parallel GA in terms of obtaining better optimal solution while reducing the computational budget.

3.2 Site Description

3.2.1 Study Area

Hang-Jia-Hu (HJH) Plain is one of most populated areas and largest economic zones in the Southeast coast of China. Located in the north of Zhejiang Province, Hang-Jia-Hu Plain is adjacent to Shanghai city and Jiangsu Province, and encompassed by Taihu Lake, Yangtz and Qiantang Rivers (as in Figure 3.1). The total area of the Plain is around $6,500\ km^2$. Within this region, surface water network accounts for approximately 10% of the land surface ([7]). Even though there seems to be an abundant surface water resource, groundwater has been an important source for domestic and agricultural water supplies due to concerns over deterioration of surface water quality and also in areas where surface water is not easily accessible.

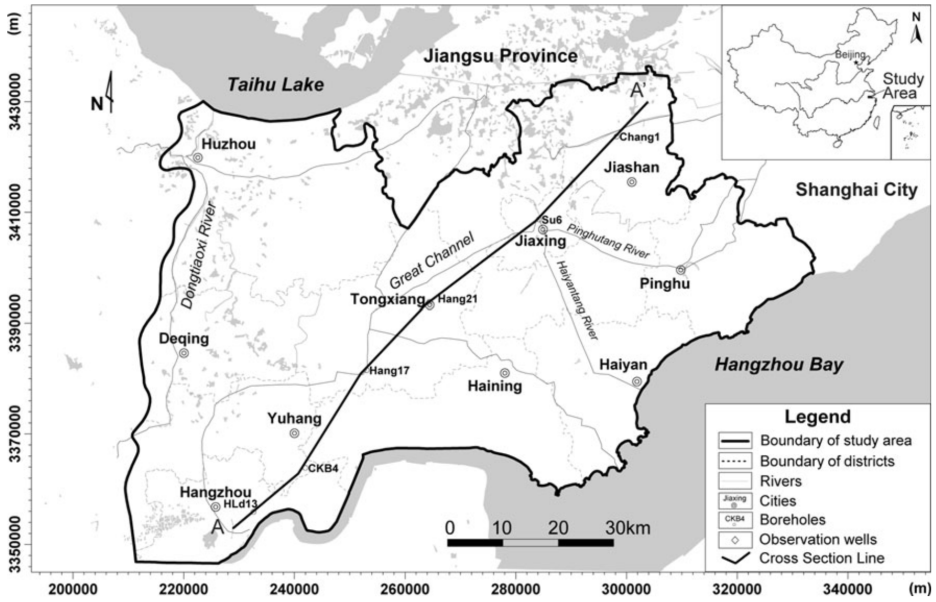


Figure 3.1: Site map of Hang-Jia-Hu area (reference: Cao et al (2013), Groundwater exploitation management under land subsidence constraint: Empirical evidence from the hangzhou-jiaxing-huzhou plain, China, Environmental Management, 51(6), 1109-1125)

3.2.2 Groundwater Exploitation situation

Extensive Groundwater extraction started in Hang-Jia-Hu Plain in 1960's. Extraction was first concentrated in urban area of Jiaxing, a large city in Hang-Jia-Hu Plain, as the yields were mainly dedicated to municipal usage. Before 1973, the extent of land subsidence was limited within the area of Jiaxing, however land subsidence increases significantly at this stage. Since 1973, groundwater extraction has grown a great deal, which was followed by land subsidence. The subsidence rates increased up to $50mm/year$ in 1988 compared to $16mm/year$ in 1973. Moreover, the affected area expanded to the suburban area of Jiaxing. After 1989, local governments began to constrain the excessive extraction. Unfortunately the rural pumping rates still increased, and land subsidence problem spread widely. Subsidence began to affect other urban areas in addition to Jiaxing. Strict government

regulation was issued to try to control subsidence in 1997. As a result, many illegally constructed wells were shut down. However the expansion of subsidence area never ceased, and Haiyan and Pinghu became two other centers of severe land subsidence. To make it worse, during the period 2001 to 2005, the surface water contamination problem increased substantially, which resulted in even more extensive groundwater pumping. Additional severe governmental groundwater exploitation prohibition measures were enacted in 2005 to limit subsidence. The substantial reduction of the total groundwater pumping gradually led to a decreasing trend of land subsidence rate over the region, but the reduction in groundwater pumping makes the water shortage problem worse ([37, 7]).

3.3 Integrated regional groundwater flow and land subsidence model

Cao et al. (2013) [7] construct an integrated regional groundwater flow and land subsidence numerical model for Hang-Jia-Hu Plain. The numerical model consists of 5 layers as mentioned before, with 1,3,5 representing confined aquifers while 2 and 4 are the aquitards. Each layer is composed of 230 rows and 320 columns. The system simulates a 12 year period transient condition in Hang-Jia-Hu Plain (1996-2007), having a total of 48 stress periods, each with a duration of 3 months. The top boundary of the model is simulated as a specific flux (groundwater recharge) and the bottom of the aquifer system is defined as a no-flow boundary. The lateral boundaries are assumed to be specific fluxes with further modification in calibration. The initial heads of aquifer are based on potentiometric maps for 1995 and initial heads for the aquitard are the interpolated results between the adjacent

aquifers. The subsidence model is calibrated based on available regional groundwater level data, land subsidence records and the seasonal withdrawal pattern of groundwater provided by Geological Survey of the Zhejiang Province.

The groundwater flow and land subsidence is modeled by MODFLOW [30] with its Subsidence (SUB) package [34]. SUB is based on one dimensional consolidation theory of Terzaghi (1925) which assumes soil deformation is vertical and the total stress/load is constant. The vertical compaction is computed based on effective stress (σ') which links groundwater withdrawal and land subsidence. Effective stress accounts for the stress loaded on soil matrix, and the remaining from the total stress is supported by pore-pressure, as described in Equation (3.1)

$$\sigma' = \sigma - u \tag{3.1}$$

where, σ' is the effective stress, σ is total stress, and u is the pore water pressure. During the withdrawal of groundwater, the piezometric head in the aquifer decreases (by assuming a constant total stress and only vertical deformation, the effective stress on the aquifer would increase, which results in the compression of soil, and as a consequence leads to land subsidence). As the decrease of hydraulic head or pore-pressure causes drop of the effective stress within the interbeds, once it reaches to a level lower than the preconsolidation stress, the deformation is considered as inelastic, otherwise, the consolidation is elastic. Preconsolidation stress/head is the lowest stress/head ever reached in the aquifer system, which is also defined as the switch between elastic and inelastic storage properties in SUB package. SUB uses different skeletal storage coefficients to handle these two situations separately. Thus, the compaction of the compacting layer $\triangle b$ and total flux

q_i^m at cell i and step m can be formulated by:

$$\triangle b = \begin{cases} S_{skv} b \triangle h, & \sigma'_{zz} \geq \sigma'_{zz(max)} \\ S_{ske} b \triangle h, & \sigma'_{zz} < \sigma'_{zz(max)} \end{cases} \tag{3.2}$$

where $\triangle b$ is the change in thickness of sediment layer, S_{sk} is the skeletal storage coefficient (S_{ske} is for elastic case, S_{skv} is for inelastic case), $\triangle h$ is the change in hydraulic head, σ'_{zz} is vertical effective stress $\sigma'_{zz(max)}$ is preconsilidation stress. At every time step, the compaction changes are computed. The expression of the total flux q (flow per unit area) into or out of elastic and inelastic skeletal storage at cell i for time-step m is:

$$q_i^m = \begin{cases} \frac{S'^m_{ke}}{\triangle t^m}(h^m - H^{m-1}) - \frac{S'_{ke}}{\triangle t^m}(H^{m-1} - h^{m-1}), & h^m > H^{m-1} \\ \frac{S'^m_{kv}}{\triangle t^m}(h^m - H^{m-1}) - \frac{S'_{ke}}{\triangle t^m}(H^{m-1} - h^{m-1}), & h^m \leq H^{m-1} \end{cases} \tag{3.3}$$

where h^m and H^{m-1} are head at end of time-step m and preconsolidation head at the end of time step $m-1$. S'_{ke} is the elastic skeletal storage coefficient and S'_{kv} is the inelastic skeletal storage coefficient.

Combining the term q_i^m with the source term W in governing equation in Equation (3.4), we are able to obtain the distribution of the hydraulic head h and as a result the change of land subsidence level ($\triangle b$) based on Equation (3.2).

The Governing Equation to simulate regional groundwater flow is described in Equation (3.4).

$$\frac{\partial}{\partial x}(K_{xx}\frac{\partial h}{\partial x}) + \frac{\partial}{\partial y}(K_{yy}\frac{\partial h}{\partial y}) + \frac{\partial}{\partial z}(K_{zz}\frac{\partial h}{\partial z}) - W = S_s\frac{\partial h}{\partial t} \tag{3.4}$$

where, K_{xx}, K_{yy}, K_{zz} are the principal components of the hydraulic conductivity tensor aligned with the x, y, z directions respectively. W is the volumetric flux per unit volume of sources (or) sinks of water, and S_s is the specific storage.

3.4 Problem formulation

The general problem we want to solve is based on Equation (3.5) as described below

$$\min_{\alpha} \; F(\alpha) = f(\alpha) + PenaltyFunction(\alpha) \tag{3.5}$$

where $f(\alpha)$ is the pure objective function, or goal to be optimized, and $PenaltyFunction$ is defined as in Equation (3.6).

$$PenaltyFunction(\alpha) = \sum_{j=1}^{TCST} max(CST_j(\alpha), 0)^2 \tag{3.6}$$

where $CST_j(\alpha)$ stands for j^{th} constraint function defined in that scenario, with $j = 1, ..., TCST$ and $TCST$ is the total number of constraints in the scenario. α is set of the solution we want to obtain. And it is defined as the fractional change in current pumping that would implement in a policy that controls subsidence. Our goal in this study takes mainly two dimensions of the HJH subsidence problem into consideration, one is the amount of pumping in order to fulfill the demand of whole region, the other is the condition of severe subsidence induced by overdraft of groundwater. The other aspects such as cost of drilling pumping wells, or different demands in each subregion are not considered within the scope of this study.

3.4.1 Zonation and decision variables

The goal of the optimization is to select groundwater extraction policies that obtain an optimal groundwater extraction plan which wins out over the current solution in terms of both amount of extraction and induced land subsidence based on the model built and calibrated by [7].

To achieve this goal, all pumping wells in the Hang-Jia-Hu plain that are at 700 well locations in the numerical model are considered. And the study area is divided into 10 zones according to the existing governmental administrative boundaries (Figure 3.2).

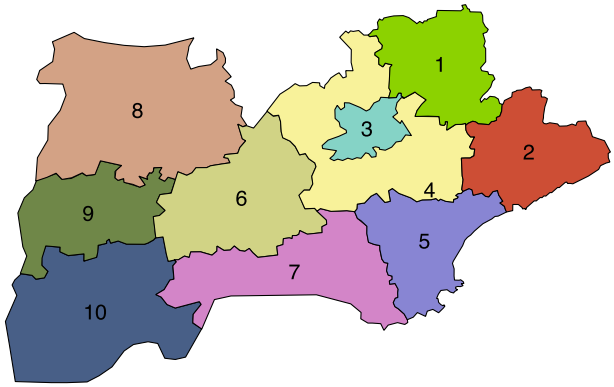


Figure 3.2: Administration Zonation of site

Land subsidence records only concentrated in seven regions, therefore wells located in those seven regions were treated as controlling wells for optimization. The other three regions have the pumping plan kept the same as the current one. Further on, we cluster well locations in each sub-region using K-means algorithm. This clustering method aims to partition N points to I clusters based on minimization of the squared distance of all points to its centroid [67]. In our case, each cluster (i) has one decision variable, which indicates a fraction α_i of current pumping rate allowed for all wells within cluster (i). In total, we have 38 clusters.

3.4.2 Formulation 1

The objective in Formulation 1 is similar to many previous studies on groundwater management ([26, 2, 67]). We want to maximize the total pumping rate over the 12

year management period (1996-2007) with constraints on magnitude of the induced land subsidence. To maintain the general trend of expansion and restriction on groundwater withdrawals, we define multiplier (α_i) as decision variable for each well cluster i. The pure objective function $f(x)$ as in Equation (3.5) is represented by $-Qsum_m(\alpha)$ in Formulation 1. The objective function can be expressed as below.

$$\min_{\alpha} \; F(\alpha) = -Qsum_m(\alpha) + Penalty_{s1}(\alpha) \qquad \alpha \in \mathbb{R}^n \tag{3.7}$$

where $Qsum_m(\alpha)$ is defined below, and the minus sign is added above to convert into a minimization problem.

$$Qsum_m(\alpha) = \sum_{i=1}^{NZ} \alpha_i \sum_{n \in Z_i} \sum_{t=1}^{NST} Q_c(n,t) \qquad \alpha \in \mathbb{R}^n \tag{3.8}$$

$F(\alpha)$ is the objective function value. $Qsum_m(\alpha)$ is the total pumping rate in the model $[L^3/T^{-1}]$. The total amount of pumping is then $30 * Qsum_m$, as each time period has a length of 30 days. NST is the total number of pumping time periods. Let Z_i be the set of indices of the wells in the i^{th} well cluster. Then $Q_c(n,t)$ is the current pumping rate of well n at time period t. The decision variables are defined as multipliers of α_i, one for each well cluster zone Z_i and for each α_i is restricted as in $5\% \leq \alpha_i \leq 300\%$. So, for example if $\alpha_i = 0.8$, then the optimized pumping rate at well i is 80% of the current pumping rate $Q_c(n,t)$ for all t. Overall, we use 38 decision variables for having 38 well clusters. NZ represents the total number of cluster zones (i.e. $NZ = 38$, $i \in (1, 2, ...38)$), and all the multipliers α are invariant among different time periods. The decision vector α is defined as $\alpha = (\alpha_1, ..., \alpha_{NZ})$.

In the objective function $F(\alpha)$, $Penalty$ stands for penalty function which is a positive value if the following constraints are violated, or zero otherwise.

The constraints are

- The sum of land subsidence value ($S_{k,NST}$) for cell grid k among all cells TNC at the end of management period (NST) in the model ($Stsum_m(\alpha)$) should not exceed the allowable sum subsidence value ($Stsum_p$)

$$Stsum_m(\alpha) = \sum_{k=1}^{TNC}(S_{k,NST}) \leq Stsum_p \tag{3.9}$$

Therefore, constraint function is defined as

$$CST_1(\alpha) = \frac{Stsum_m(\alpha) - Stsum_p}{Stsum_p} \tag{3.10}$$

- Area of subsidence (AS) where subsidence value over 50mm at the end of management period NST) cannot exceed the permissible subsidence affected area (AS_{p1})

$$AS_{S\geq 50mm,NST}(\alpha) \leq AS_{p1} \tag{3.11}$$

Therefore, constraint function is defined as

$$CST_2(\alpha) = \frac{AS_{S\geq 50mm,NST}(\alpha) - AS_{p1}}{AS_{p1}} \tag{3.12}$$

- Area of subsidence (AS) where subsidence value over 60mm at the end of management period NST) cannot exceed the permissible subsidence affected area (AS_{p2})

$$AS_{S\geq 600mm,NST}(\alpha) \leq AS_{p2} \tag{3.13}$$

Therefore, constraint function is defined as

$$CST_3(\alpha) = \frac{AS_{S\geq 600mm,NST}(\alpha) - AS_{p2}}{AS_{p2}} \tag{3.14}$$

The $Penalty_{s1}$ term in Eqn (3.7) contains constraint functions that are scaled and squared in order to converge fast to the feasible region. Its form is shown as in Equation (3.15), with weights for each constraint assigned as w_{s1}.

$$Penalty_{s1}(\alpha) = w_{s1} * (max(CST_1(\alpha), 0)^2 + max(CST_2(\alpha), 0)^2 + max(CST_3(\alpha), 0)^2) \tag{3.15}$$

3.4.3 Formulation 2

The aim of second formulation is to minimize the land subsidence condition with a constraint that the groundwater extraction must exceed a minimum amount. This formulation needs to be used under the circumstances when the goal of project is subsidence protection. The mathematical expression of the objective function is shown below. The pure objective function $f(\alpha)$ as in Equation (3.5) is represented by $AS_{s\geq 500mm,NST}(\alpha)$ in Formulation 1.

$$\min_{\alpha} \; F(\alpha) = AS_{s\geq 500mm,NST}(\alpha) + Penalty_{s2}(\alpha) \tag{3.16}$$

where $Penalty_{s2}$ is defined in Eqn. (4.1) and $AS_{s\geq 500mm,NST}$ is area of subsidence (AS) where subsidence value is over $500mm$ at the end of management period NST The constraints are as follows

- A minimal total pumping required

$$Qsum_m = \sum_{i=1}^{NZ} \alpha_i \sum_{n\in Z_i} \sum_{t=1}^{NST} Q_c(n,t) \geq Qsum_p \tag{3.17}$$

  So, constraint function CST_1 is defined as

$$CST_1(\alpha) = \frac{Qsum_p - Qsum_m(\alpha)}{Qsum_p} \tag{3.18}$$

- The summation of subsidence level computed in the model at the end of management period T at all cells k (overall TNC cells) denoted as $Stsum_m$ cannot exceed the permissible total subsidence value ($Stsum_p$)

$$Stsum_m(\alpha) = \sum_{k=1}^{TNC} (S_{k,NST}) \leq Stsum_p \tag{3.19}$$

  So, constraint function CST_2 is defined as

$$CST_2(\alpha) = \frac{Stsum_m(\alpha) - Stsum_p}{Stsum_p} \tag{3.20}$$

The $Penalty_{s2}$ term in Eqn. (3.17) contains constraints below:

$$Penalty_{s2}(\alpha) = w_{s2} * (max(CST_1(\alpha), 0)^2 + max(CST_2(\alpha), 0)^2) \tag{3.21}$$

3.4.4 Formulation 3

The third formulation represents the case of extraction with all deep wells, i.e. we simulate the situation that all wells are drilled to the deepest aquifer available. The objective function is similar to that in Formulation 1, which is to maximize the total pumping, but the decision variables here are multipliers assigned to all deep pumping wells. We denote the decision variable as α^d, where $\alpha^d * Q_c(n,t)$ is the amount of pumping from the deepest aquifer at locations n, with also limitation on each pumping well i, as $5\% \leq \alpha_i^d \leq 300\%$. There are three constraints same as in Formulation 1: one is to limit the summation of cumulative subsidence at all MODFLOW cell at end of management period; the others are the limitation of the areas of subsidence (AS) where subsidence value over $500mm$ and over $600mm$ at the end of project. To avoid repetition, the specific mathematical formulation is not shown here, as it is the same as in Formulation 1 except for a change of variable (i.e. α^d replaces α).

3.5 Optimization Algorithms

3.5.1 parallel GA

Genetic Algorithm [74] is a widely known algorithm, which is based on idea of mimicking the search process of natural selection. It solves an optimization prob-

lems through evolving the best solution from an initial set of random guess. During the evolution, different strategies can be applied. One strategy is "crossover", a recombination of the "genes" (elements of the decision variables); therefore the offsprings can inherit the merits of "genes" from the parents. Offsprings can also mutate, as "mutation" is a strategy to add diversity into the population, Then the pairs of parents for the next generation are chosen based on a procedure that gives a higher likelihood of selection to parents with a higher objective function value. As the number of generation increases, the population evolves to the solution. In serial GA, the fitness values are computed one at a time, while for Parallel GA, fitness values are computed simultaneously on multiple computing cores.

In this work, we employed Distributed Evolutionary Algorithm in Python (DEAP) module [19] and built the parallel real GA. DEAP is a python toolbox for Evaluation Strategy combined with MPI. In the toolbox, we chose two point crossover, uniform mutation and tournament selection. Algorithm Parameters (crossover probability and mutation rate and size of tournaments) are set as same as default values in MATLAB genetic algorithm Toolbox.

3.5.2 parallel DYCORS

DYCORS (Dynamic Coordinate search using Response Surface models) is a surrogate based optimization method that specializes in dealing with large number of controlling variables [60]. It is an efficient algorithm suitable for high-dimensional, expensive, and black-box problems, so that we can solve for physically-based subsidence model and provide optimal pumping strategy for controlling subsidence with limited simulations required. The main idea in DYCORS algorithm is that we interactively select random trial points with the aid of updated response surface in

each iteration. The response surface is chosen as Radial Basis Function. And as to trial point selection, we use the strategy of decreasing the probability of perturbing a coordinate as the algorithm proceeds on to reach the computational budget. This strategy is incorporated in Regis (2013) [60] for surrogate-based optimization and it was suggested earlier in for heuristic search [71].

3.5.3 parallel DYSOC

We introduce here a new algorithm DYSOC targeted at problems with expensive constraints. The idea is to build response surfaces $S_i^{cts_j}$, one for each of the constraint function $CST_j(x)$ in addition to one surrogate surface S_i^{obj} only for objective function. A similar concept called ConstrLMSRBF has been applied on Stochastic Radial Basis Function algorithm by [56], in which additional RBF response surfaces for constraint are built to guide selection of next point to evaluate. DySOC combines features of DYCORS and ConstrLMSRBF. Here we use pure objective function $f(x)$ instead of $F(x)$ in Equation (3.5) to build the surrogate surface S_i^{obj}. To be more precise, a capped objective function as $f_{capped}(x)$ is used in each iteration i as in Equation (4.8) (i.e. $S_i^{obj}(x)$ is based on points in $\mathcal{B}_i = \{(x, f_{capped}(x)) : x \in \mathcal{A}_i\}$).

$$f_{capped}(x) = \min(f(x), \text{median}_{n \in \mathcal{A}_i} f(x_n)) \tag{3.22}$$

Since our study is a minimization problem, points with large evaluated value of $f(x)$ are of less interest, therefore, this capped function helps focusing the search for the minimal solution and avoids numerical instabilities. The response surface for a constraint can facilitate the search for feasible domain or at least for domain of points with small violation in constraint functions. To be noted here is that

in our study, the objective function used in DYSOC is in form of $F(x)$ which is the summation of the pure objective function (i.e. $f(x)$) and a penalty for the constraints (i.e. $PenaltyFunction(x)$), while surrogate surface S^{obj} only approximates $f(x)$, It is different from previous studies in Regis (2011) [56], as [56] used $f(x)$ as both objective function and S^{obj}.

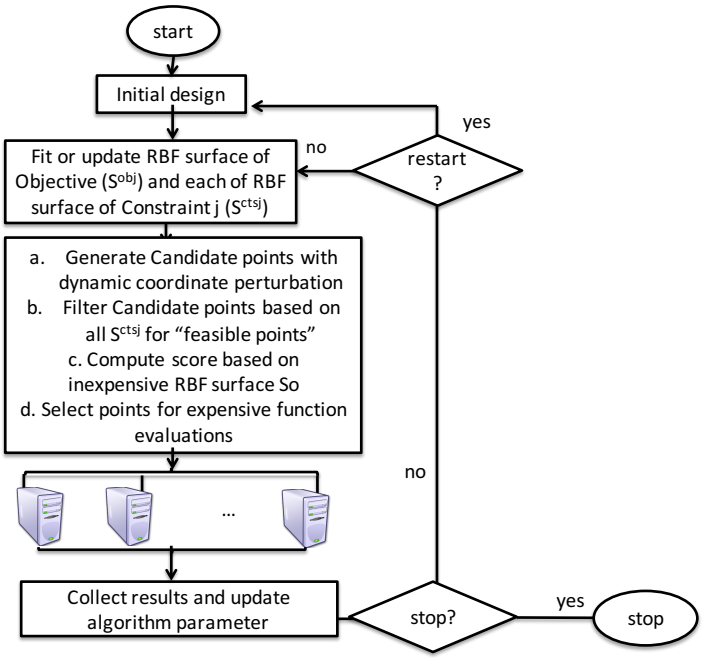


Figure 3.3: Framework of DYSOC

As described in Algorithm 1, after initialization from Step 1 to 3, we then generate candidate point Ω_i. Same as in DYCORS, Ω_i is found by perturbing the best solution x_{best} of the objective function $F(x)$ by using the Equation (3.23). In [56], a uniform perturbation is used for generating the candidate points Ω_i instead. Details of how to generate Ω_i in DYSOC are described in Function box **Candidate_Point_Selection** in Step 9. We define $\mathcal{P}_i$ as a set of potentially feasible points with a size of N at each iteration i. Then, we generate the set $\mathcal{P}_i$ by filtering for "least violated" points based on $S_i^{cts_j}$ from candidate points

Ω_i in Step 10 (i.e. N points in $\mathcal{P}_i$ are N points with smallest values of function $\sum_j max(0, S_i^{cst_j})$ in candidate points set Ω_i).

$$p^{select}(i) = p_0^{select}[1 - \frac{log(i - m + 1)}{log(I_{max} - m)}] \tag{3.23}$$

where i is the number iteration number, p_0^{select} is an initial probability, I_{max} is the maximum number of iterations, m is the number of evaluations in initial experiment design.

Algorithm 1: DYSOC

Input: Initial experimental design, Sample point strategy, Surrogate model for objective, Surrogate model for constraints,Stopping criterion, Restart criterion

Output: Best solution and its corresponding function value

1 Generate an initial experimental design $\mathcal{A}_0 := \mathcal{I}$;
2 Evaluate F(x) for the points x in $\mathcal{A}_0$ in the experimental design;
3 Build a Surrogate model $S_0^{obj}(x)$ for $\mathcal{B}_0^{obj} = \{(x, f_{capped}(x)) : x \in \mathcal{A}_0\}$ and surrogate models $S_0^{cst_j}$ for $\mathcal{B}_0^{cst_j} = \{(x, CST_j(x)) : x \in \mathcal{A}_0\}$ for each constraint j;
4 **repeat**
5 **if** *Restart criterion met* **then**
6 Reset the Surrogate model and the Sample point strategy;
7 **go to** 3;
8 **end**
9 Generate multiple Candidate points Ω_i using **Candidate_Point_Selection**;
10 Filter the Candidate points Ω_i for a set of least violated points $\mathcal{P}_i$ predicted by $S_{i-1}^{cst_j}$ for all j, i.e. for $x \in \mathcal{P}_i$ are $x \in \Omega_i$ which have small values of $\sum_j max(0, S_{i-1}^{cst_j})$;
11 Select new points $\mathcal{D}_i$ to evaluate among Candidate points $\mathcal{P}_i$ based on metric function applied to **Selection_Metrics**$(\mathcal{P}_i, S_{i-1}^{obj}(x), S_{i-1}^{cst_j}(x))$;
12 Evaluate the points $\mathcal{D}_i$ generated using all computational resources;
13 Update the Surrogate model $S_i^{obj}(x)$ for $\mathcal{B}_i^{obj} = \{(x, f_{capped}(x)) : x \in \mathcal{A}_i = \mathcal{A}_{i-1} \bigcup \mathcal{D}_i\}$ and surrogate models $S_i^{cst_j}$ for $\mathcal{B}_i^{cst_j} = \{(x, CST_j(x)) : x \in \mathcal{A}_i = \mathcal{A}_{i-1} \bigcup \mathcal{D}_i\}$ for each constraint j ;
14 **until** *Stopping criterion not met*;

Then it follows by the Step 11, in which we choose evaluation points $\mathcal{D}_i$ from

function $\Omega_i = \textbf{Candidate\_Point\_Selection}(x_{best}, p^{select})$
(a) Select coordinates to perturb. Generate K uniform random numbers ω_1, ... ,ω_K in [0,1] for K dimensional problem. Let $I_{perturb} := \{k : \omega_k < p^{select}\}$, with p^{select} defined in Eqn (3.23). If $I_{perturb} = \emptyset$, then select j uniformly at random from $\{1, ..., K\}$ and set $I_{perturb} = \{j\}$.
(b) Generate trial point. Generate $y_{i,j}$ by $y_{i,j} = x_{best} + z$ in i^{th} iteration for j^{th} dimension, where $z^{(ipb)} = 0$ for all $ipb \in I_{perturb}$ and $z^{(ipb)}$ is a realization of a normal random variable with mean 0 and standard deviation σ_i in i^{th} iteration for all $ipb \in I_{perturb}$.
(c) Ensure trial point is in domain. Let $\mathcal{M}$ defines the decision domain for decision variable. If $y_{i,j} \notin \mathcal{M}$, then replace it by a point in $\mathcal{M}$ obtained by performing successive reflection of $y_{i,j}$ about the closest point on the boundary of $\mathcal{M}$.

$\mathcal{P}_i$ based on metric function applied to distance information with points in $\mathcal{A}_i$ and a weighted sum of values predicted by response surfaces including S_i^{obj} and all $S_i^{cts_j}$ as described in Function box **Selection_Metrics**. After expensive computation on all points of $\mathcal{D}_i$ by all computing resources, the evaluated solutions $f_{capped}(x)$ and $CST_j(x)$ of $\mathcal{D}_i$ are then used to update the corresponding surrogate surfaces.

function $\mathcal{D}_i = \textbf{Selection\_Metrics}(\mathcal{P}_i, S_i^{obj}(x), S_i^{cst_j}(x))$
(a) *(Estimate Function Value of Candidate Points)* For each $x \in \mathcal{P}_i$, compute ζ_i which a function of weighted summation on values evaluated on $S_i^{obj}(x)$, $S_i^{cst_j}(x)$, i.e. $\zeta_i = w_1 * S_i^{obj}(x) + w_2 * \sum_j^{TCST} S_i^{cst_j}(x)$;
(b) *(Determine Minimum Distance from Previously Evaluated Points)* For each $x \in \Omega_i$, compute $\Delta_i(x) = \min_{1 \le n \le i} \|x - x_n\|$. (Here,$\|\cdot\|$ is the Euclidean norm on $\mathbb{R}^D$) ;
(c)*(Compute Weighted Score and Select Next Evaluation Point)* For each $x \in \Omega_i$, compute $\mathcal{W}_n(x) = \theta_1(\zeta_i(x) - \zeta_i^{min}) + \theta_2(\Delta_i(x) - \Delta_i^{min})$, and find x_{i+1} as the point in Ω_i that minimizes $\mathcal{W}_i$.
More details of θ_1 and θ_2 (which include normalization functions) can be found in [57].

Both DYCORS and DYSOC work in their paralleled version, which means that when evaluating the expensive objective functions including the constraint functions, we use several computing cores to conduct the computation. Therefore, in each iteration, supposing that we use p cores, then p number of evaluated inputs

are updated in order to reduce the computational time required. In the p points are selected based on method in Regis (2009) [59]. Here in this study, p is set as 16, thus we are using 16 cores simultaneously.

3.6 Results and Discussion

3.6.1 Optimal solutions

Different formulations in this study provide different optimal policies that address the subsidence issue in Hang-Jiang-Hu area from different view points. The policies are presented as follows, Table 3.1 summarizes the volumes of groundwater extracted during the twelve years' management period for each of the seven zones in three different formulations. Figure 3.4-3.7 and Figure 3.8-3.11 show the status of land subsidence and status of groundwater level respectively in optimal results of three formulations compared to the current subsidence status at the end of management period. Figure 3.12 provides the distribution of the amount of pumping in different zones in the original solution and the optimal solutions in three different formulations.

For Formulation 1 with the goal to maximize the pumping without introducing severe land subsidence, the optimal solution shows that we overall increase around 0.7% of total pumping, while maintaining the averaged subsidence values the same and reducing the area effected by severe subsidence (subsidence value $> 50cm$ or $> 60cm$). Due to a different spatial distribution of pumping, the induced land subsidence area has been redistributed and the areas with critical land subsidence have been reduced.

Table 3.1: Optimal pumping strategies in different zones in three different formulations

Zone ID	Current pumping	Optimal in Formulation 1	Optimal in Formulation 2	Optimal in Formulation 3
1	45,431,130	80,161,860	52,186,260	51,753,540
2	66,971,730	46,342,230	35,207,010	23,362,650
3	15,009,210	30,313,830	4,120,050	4,782,030
4	43,938,780	50,703,660	67,890,120	74,320,320
5	46,468,050	13,993,170	57,199,590	41,013,840
6	28,582,920	31,342,080	16,795,500	63,089,430
7	43,096,530	38,656,320	56,115,450	100,168,380
Total (m^3)	289,498,350	291,513,150	289,513,980	358,490,190

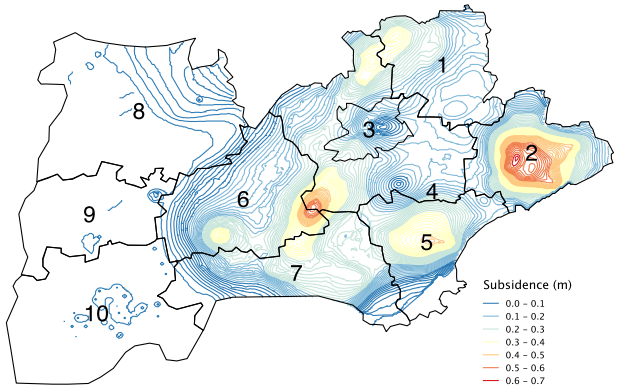


Figure 3.4: Current *in-situ* distribution of Subsidence level in Layer 1 at year 2007

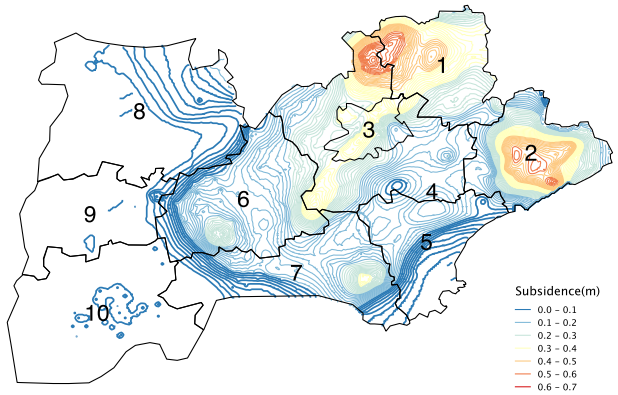


Figure 3.5: Distribution of Subsidence level in Layer 1 at year 2007 in Optimal solution in Formulation 1

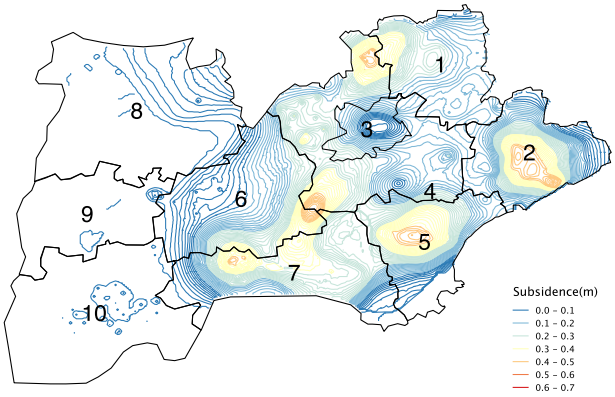


Figure 3.6: Distribution of Subsidence level in Layer 1 at year 2007 in Optimal solution in Formulation 2

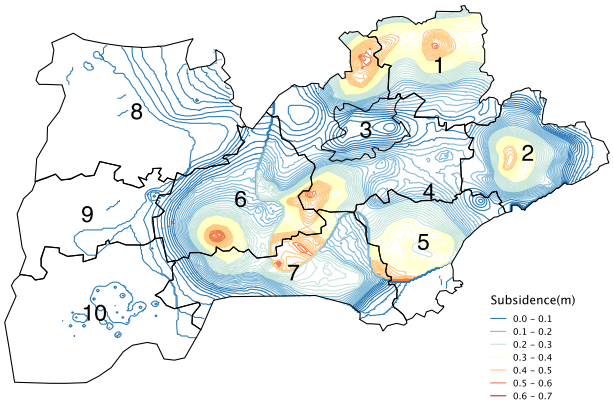


Figure 3.7: Distribution of Subsidence level in Layer 1 at year 2007 in Optimal solution in Formulation 3

Compared to current results in Figure 3.4, solution of Formulation 1 in Figure 3.5 shows that areas which are largely affected by subsidence such as Zone 2 have reduced total pumping in the optimal result, and redistribution of pumping alleviates severe land subsidence areas in Zone 5. Table 3.1 shows that places that get less influence of subsidence such as the northeastern areas as in Zone 1 and Zone 3 are able to allow more pumping. The subsidence centers have been shifted from the central to the northeastern part in Hang-Jia-Hu area in optimal solution, as we increase pumping in Zone 3,4,6 as shown in Figure 3.12. Table 3.2 summarizes the

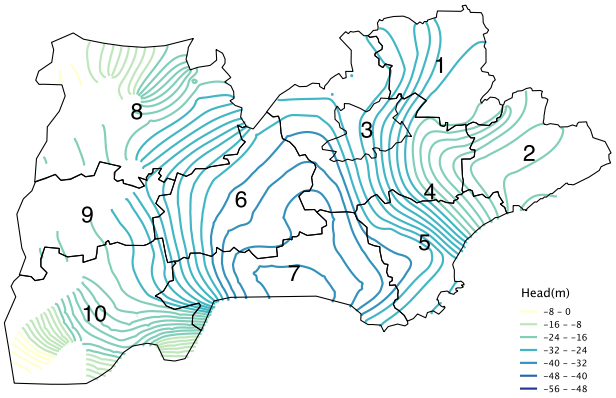


Figure 3.8: Current *in-situ* distribution of groundwater head in Layer 1 at year 2007

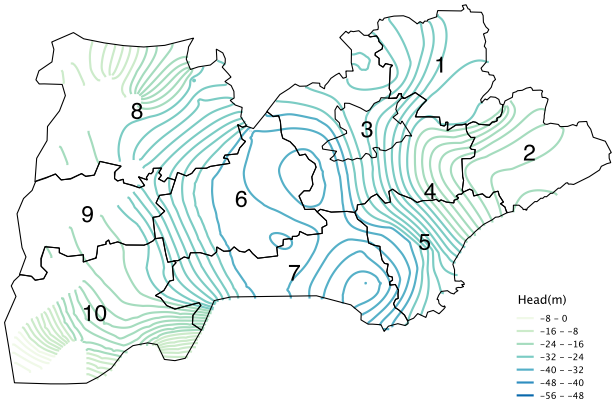


Figure 3.9: Distribution of groundwater head in Layer 1 at year 2007 in Optimal solution in Formulation 1

induced land subsidence status, in which we can see all constraints are satisfied. The Figure 3.9 of water level of Layer 1 shows that there are high water heads in the central part of this area compared to current results, as in Zone 6 and Zone 7.

Table 3.2: Induced land subsidence from optimal pumping results in three different formulations, F1, F2, F3 stands for Formulations 1,2,3

	current	F1	F2	F3
$AS_{S\geq 500mm,NST}$ (km^2)	94.08	87.68	0	48.16
$AS_{S\geq 600mm,NST}$ (km^2)	5.76	0	0	0
mean subsidence at 2007 (cm)	6.72	6.72	6.71	6.72

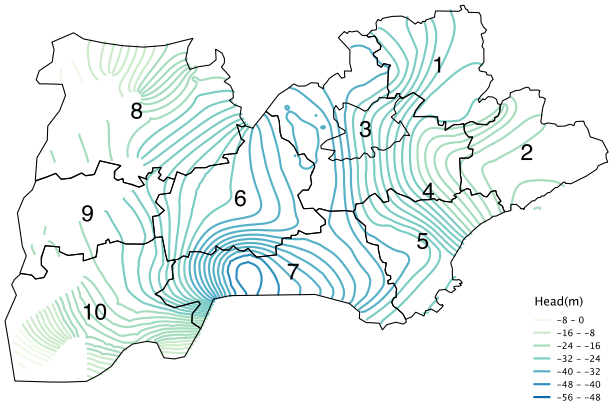


Figure 3.10: Distribution of groundwater head in Layer 1 at year 2007 in Optimal solution in Formulation 2

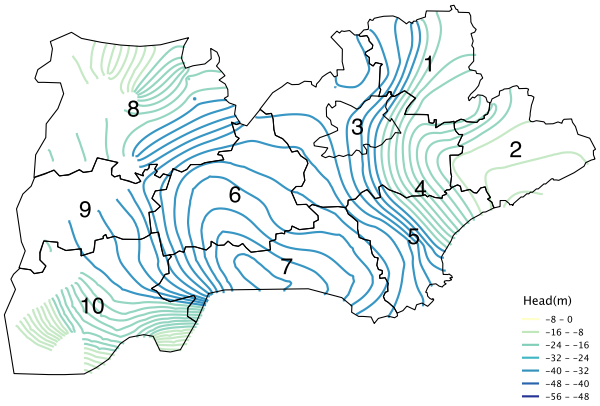


Figure 3.11: Distribution of groundwater head in Layer 1 at year 2007 in Optimal solution in Formulation 3

For the objective to minimize the land subsidence in Formulation 2, the optimal result shows that no area has cumulative land subsidence value greater than 50cm, meanwhile both the minimal demand of pumping and the total sum of cumulative subsidence constraints are satisfied. From the Figure 3.6, the distribution of optimal cumulative land subsidence also shows an reduced subsidence condition, which contains less severe subsidence areas in the majority part in Hang-Jia-Hu area. In Figure 3.10, it can be seen that the water head of Layer 1 drops in Zone 7 compared to current situation.

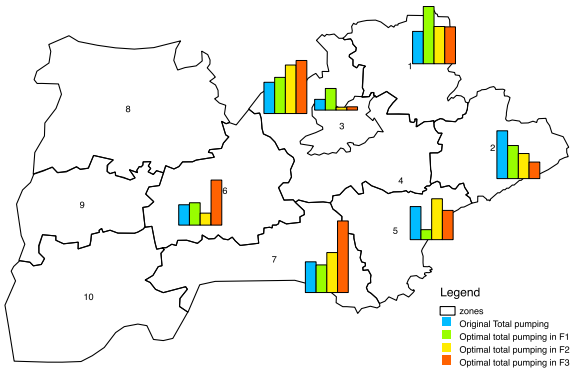


Figure 3.12: Total groundwater pumping in optimal solution in three formulations and the original pumping in different zones

In Formulation 3, we investigate the situation of pumping from deep wells, in which the depths of all wells are increased to the deepest aquifer. The optimal solution in Table 3.1 shows that we are able to achieve 23.8% more the pumping with less effect area of more than 50cm subsidence and no area with more than 60cm subsidence, and we obtain almost the same averaged subsidence value (level) in the whole region. The cumulative subsidence distribution at the end of 2007 is in Figure 3.7, which shows a more splitted distribution of areas with subsidence level greater than 50cm, and the hydraulic head level in Figure 3.11 shows that there are higher hydraulic heads of Layer 1 in general in Hang-Jia-Hu area.

Figure 3.12 shows that with different redistributions of the groundwater pumping, we are able to achieve different goals. The distributions of pumping in the optimal solution in Formulation 1 and 3 show that we should extract more water from the less severe subsidence affected area such Zone 1 and Zone 4, and control extraction on severely affected area such as Zone 2. Formulation 2 (which is the case with constraint on the total groundwater demand) illustrates that if we need to fulfill the demand of each zone in original solution, we need to ship water out

of zones such Zone 1,4,5,7 where optimal solution yield smaller pumping amount than the demand into Zone 2,3,6 where demands are higher than the solution in optimal results (assuming demands are current amount of pumping). But the demand can change as a result of the water availability and the shipping cost is not considered in our optimization.

3.6.2 Comparison of different algorithms

Comparison of different algorithms is shown in Figure 3.13 - 3.15. To ensure a fair comparison, parallel DYCORS (p-DYCORS) and parallel DYSOC (p-DYSOC) start with the same initial sampling design, and the initial population in parallel GA (p-GA) has the worst individual replaced by the best point we find in the initial sampling design of p-DYCORS (or p-DYSOC, since they are the same). For each comparison, we conduct 20 trials with different initial designs, and 600 function evaluations are conducted for each algorithm.

The progress graphs (i.e. Figure 3.13 - 3.15) illustrate the evolution of the best averaged result obtained so far over 20 trials along the optimization process. As we convert all three formulations to minimization problems, the lowest curves in all progress graphs thus represent the algorithm with the best efficiency. As we can see, p-GA shows the worst averaged performance, and p-DYSOC performs the best in all formulations. The horizontal level in Formulation 1 represents amount of pumping in current in-situ policy. As shown in Figure 3.13, we can see that both p-DYSOC and p-DYCORS can obtain better solution than the current policy in their averaged performance, whereas averaged p-GA cannot reach the solution better than current policy. Figure 3.13 plots the results starting from 100 function evaluations and zooms into the function values ranging from $-9.7*10^6$ to $-9.5*10^6$

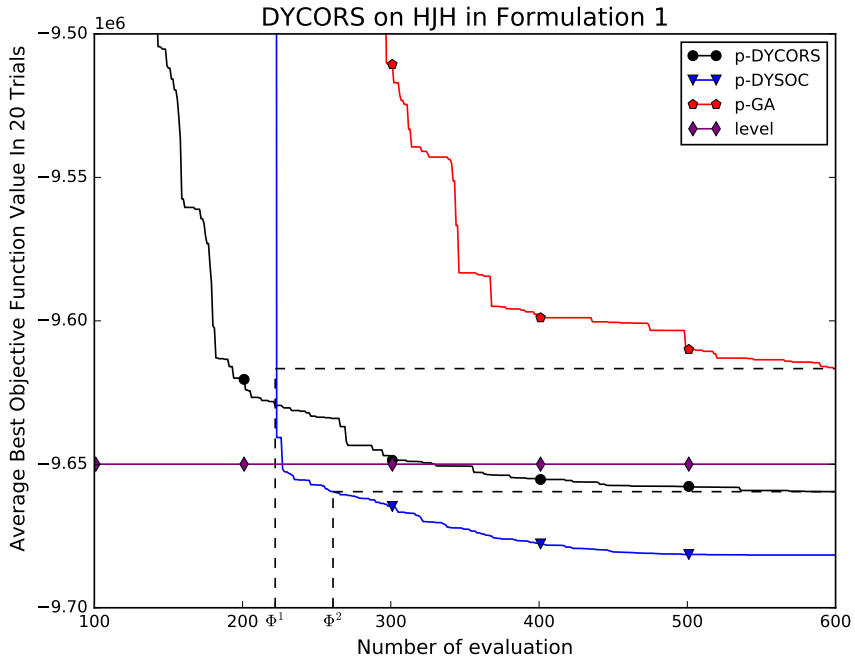


Figure 3.13: Averaged best Objective function $F(x)$ in Equation (3.7) among 20 trials against number of evaluations (from 100 to 600) comparing p-DYCORS, p-DYSOC and p-GA in Formulation 1 for Hang-Jia-Hu (HJH) region. Here Formulation 1 is converted to a minimization problem, so the smaller the value is, the better the algorithm performs. The purple line at -9.65*10^6 gives the objective value of the current solution, and it shows that p-DYCORS and p-DYSOC outperforms the current solution. $\Phi^1 = \Phi(DYSOC, GA, 600)$ and $\Phi^2 = \Phi(DYSOC, DYCORS, 600)$

to clearly illustrate the difference between results obtained with three algorithms. The results beyond Figure 3.13contain that the averaged solution of p-GA at 200 function evaluations is around $2 * 10^8$ which is way larger than $-4 * 10^6$ obtained by p-DYSOC and $-9.6 * 10^6$ obtained by p-DYCORS. And p-DYSOC catches up with p-DYCORS after 200 function evaluations, and obtained a result better than p-DYCORS after around 220 function evaluations. As to Formulation 2, averaged solution at 200 function evaluations of p-GA is 730 (out of Figure 3.14) which is also much larger than 194 obtained by p-DYSOC and 267 obtained by p-DYCORS. Figure 3.14 shows that the averaged optimal solution of p-DYSOC outperforms the

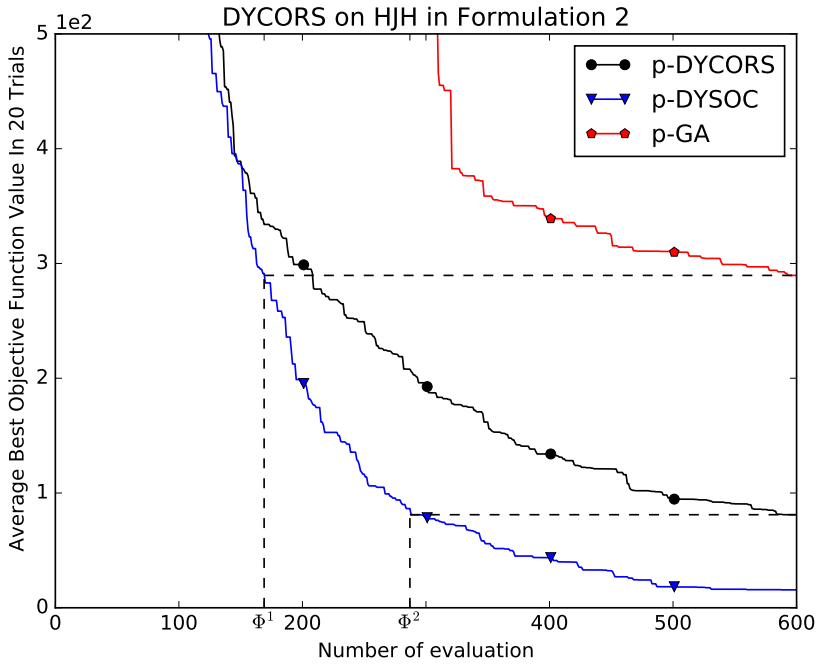


Figure 3.14: Averaged best Objective function $F(x)$ in Equation (3.16) among 20 trials against number of evaluations (up to 600) comparing p-DYCORS, p-DYSOC and p-GA in Formulation 2. Here Formulation 2 is a minimization problem, so the smaller the value is the better the algorithm performs. $\Phi^1 = \Phi(DYSOC, GA, 600)$ and $\Phi^2 = \Phi(DYSOC, DYCORS, 600)$

other two algorithms at any function evaluation, as its curve stays lowest among all. In Formulation 3, p-DYSOC has a superior averaged performance over both p-GA and p-DYCORS. And averaged performance of p-GA find solution with objective smaller than $-0.9 * 10^7$ after around 400 function evaluations which is much slower than p-DYCORS and p-DYSOC.

A proper way to compare two algorithms A_1 and A_2 is to compare the speedup, i.e. what is the ratio of computing time required for both algorithms to reach the same value. We define $\Phi(A_1, A_2, N)$ as the number of evaluation as required by algorithm A_1 to get the same average solution A_2 gets in N evaluations. The value of $\Phi(A_1, A_2, 600)$ is shown in Fig 3.13 - 3.15 for $\Phi^1 = \Phi(DYSOC, GA, 600)$ and

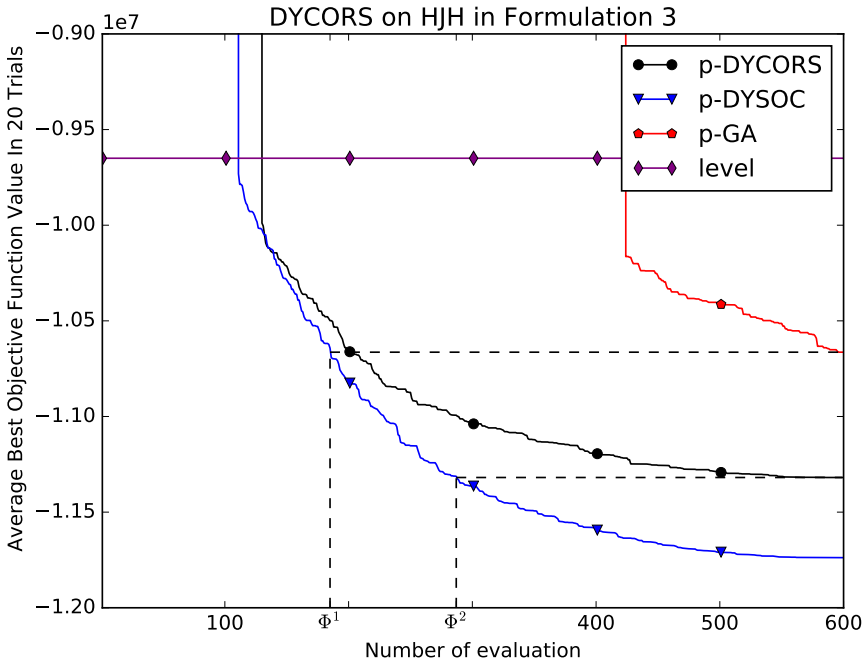


Figure 3.15: Averaged best Objective function $F(x)$ similar as in in Equation (3.7) with a change variable from α to α^d among 20 trials against number of evaluations (up 100 to 600) comparing p-DYCORS, p-DYSOC and p-GA in Formulation 3, here Formulation 3 is converted to a minimization problem, so the smaller the value is the better the algorithm performs. The purple line at $-9.65{*}10^6$ gives the objective value for the current solution. $\Phi^1 = \Phi(DYSOC, GA, 600)$ and $\Phi^2 = \Phi(DYSOC, DYCORS, 600)$

$\Phi^2 = \Phi(DYSOC, DYCORS, 600)$. Then speed up is defined as in Eqn (3.24)

$$\text{Speedup} = S_{up}(A_1, A_2, N) = \frac{N}{\Phi(A_1, A_2, N)} \tag{3.24}$$

We base the computation of $\Phi(A_1, A_2, N)$ on the mean values over multiple trials since the algorithm is stochastic.

Results of $\Phi(A_1, A_2, N)$ and $S_{up}(A_1, A_2, N)$ are summarized in Table 3.3. In Table 3.3, it shows that in Formulation 1, p-DYSOC gets the solution in 222 functions that it takes GA 600 function evaluations to get, so the speed up of p-DYSOC over GA (i.e. $S_{up}(DYSOC, GA, 600)$) is $\frac{600}{222} = 2.7$. We can see that in Table 3.3, p-DYSOC can obtain speed up of $2.7 \sim 3.5$ compared to p-GA at

Table 3.3: Number of function evaluations ($\Phi(A_1, A_2, N)$) required for algorithm A_1 to reach to the same averaged solution algorithm A_2 gets in N evaluations among 20 trials. And Speed up ($S_{up}(A_1, A_2, N)$) comparing A_1 and A_2. Here A_1 is DYSOC and A_2 is DYCORS or GA.

Formulation	Benchmark of 600 evaluations	Measures
1	DYCORS	Φ(DYSOC,DYCORS,600) = 261
		S_{up}(DYSOC,DYCORS,600) = 2.30
	GA	Φ(DYSOC,GA,600) = 222
		S_{up}(DYSOC,GA,600) = 2.70
2	DYCORS	Φ(DYSOC,DYCORS,600)=287
		S_{up}(DYSOC,DYCORS,600)=2.09
	GA	Φ(DYSOC,GA,600) = 169
		S_{up}(DYSOC,GA,600)= 3.55
3	DYCORS	Φ(DYSOC,DYCORS,600)=287
		S_{up}(DYSOC,DYCORS,600)= 2.09
	GA	Φ(DYSOC,GA,600) =185
		S_{up}(DYSOC,GA,600)= 3.25

its results after 600 function evaluations. By comparing to averaged results of p-DYCORS, p-DYSOC can achieve speed up of $2.09 \sim 2.3$. As we can see, the results show a great reduction in the computation effort required for p-DYSOC compared to the other two algorithms.

Due to stochastic characteristic of the algorithms we compare, an statistical analysis is made in order to evaluate the consistency in the performance of algorithms among different trials. One way is to look at statistics in results among trials after specific function evaluations. Table 3.4 shows the mean and standard deviation of p-GA, p-DYCORS and p-DYSOC, each after 300, 400, 500, 600 function evaluations. Noted here all problems are converted into minimization problems. In Formulation 1, results show that the smallest averaged optimal value and the smallest standard deviation is provided by p-DYSOC at each of the four stages of optimization (i.e. 300, 400, 500, 600). As to Formulation 2, p-DYSOC obtains the

Table 3.4: Statistics of results by different algorithms DYCORS, DYSOC and GA among 20 trials at different number of function evaluations (i.e. N as in "Neval"). Here results are presented in its objective in optimization. So in all formulations, the smaller the "mean" value is, the better the algorithm performs. The empty cell for GA means that GA cannot obtain a feasible result with either 300 or 400 function evaluations in Formulation 3

F	Algorithm	Stats	300eval	400eval	500eval	600eval
1	DYSOC	mean	-9.67×10^6	-9.68×10^6	-9.68×10^6	-9.68×10^6
		std	2.26×10^4	2.14×10^4	2.24×10^4	2.23×10^4
	DYCORS	mean	-9.65×10^6	-9.66×10^6	-9.66×10^6	-9.66×10^6
		std	2.46×10^4	2.25×10^4	2.24×10^4	2.27×10^4
	GA	mean	-9.51×10^6	-9.60×10^6	-9.61×10^6	-9.62×10^6
		std	2.48×10^5	1.47×10^5	1.38×10^5	1.25×10^5
2	DYSOC	mean	78.6	43.7	18.3	15.6
		std	71.3	56.5	27.7	24.7
	DYCORS	mean	196	134	94.8	81.1
		std	179	134	123	119
	GA	mean	554	339	310	290
		std	768	278	276	272
3	DYSOC	mean	-1.14×10^7	-1.169×10^7	-1.17×10^7	-1.17×10^7
		std	2.06×10^5	1.64×10^5	1.51×10^5	1.46×10^5
	DYCORS	mean	-1.10×10^7	-1.12×10^7	-1.13×10^7	-1.13×10^7
		std	2.79×10^5	1.99×10^5	1.17×10^5	1.04×10^5
	GA	mean	-	-	-1.04×10^7	-1.07×10^7
		std	-	-	5.61×10^5	4.38×10^5

smallest averaged optimum and the smallest standard deviation. In Formulation 3, the lowest mean value goes with p-DYSOC, but its standard deviation of values at the end of run is slightly higher than p-DYCORS. p-GA performs the worst as at the beginning of run as it cannot find feasible solution, and at the end, its solution obtained is higher (i.e. worse) than the other two algorithms and also with large variance. Overall, p-GA is the worst both in terms of averaged performance and its consistency in providing the solutions among different trials.

Box plots of Fig.3.16 - Fig.3.18 show that in results of p-GA has large variation comparing agains both p-DYCORS and p-DYSOC in all formulations. And com-

paring between p-DYCORS and p-DYSOC, we have smaller variation in results of p-DYSOC after both 300 and 600 function evaluations in Formulation 1, 2 and 3.

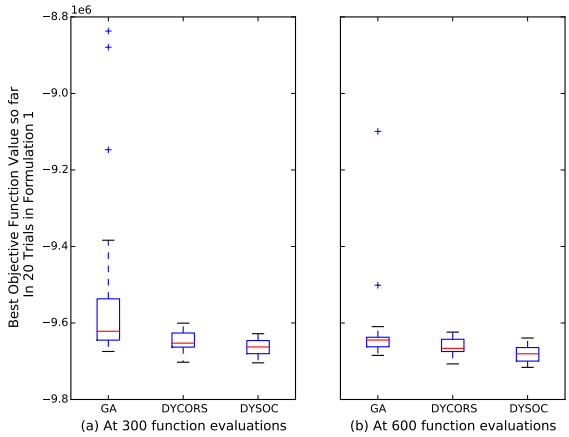


Figure 3.16: Box plot of results among 20 trials in Formulation 1.

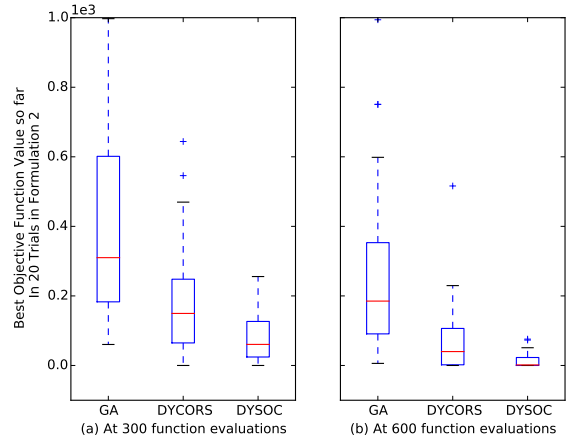


Figure 3.17: Box plot of results among 20 trials in Formulation 2.

A statistical test has been also conducted on the results after 600 function evaluations for pairs of algorithms in different formulations. We use Mann-Whitney Rank Sum Test which is a non-parametric test on dataset with sample size of 20 (as we have 20 trials). It shows in Table 3.5 that at significance level of 5%, p-DYCORS performs significantly better than p-GA in Formulation 2 and Formulation 3, but not in Formulation 1. And p-DYSOC is significantly better than the other two

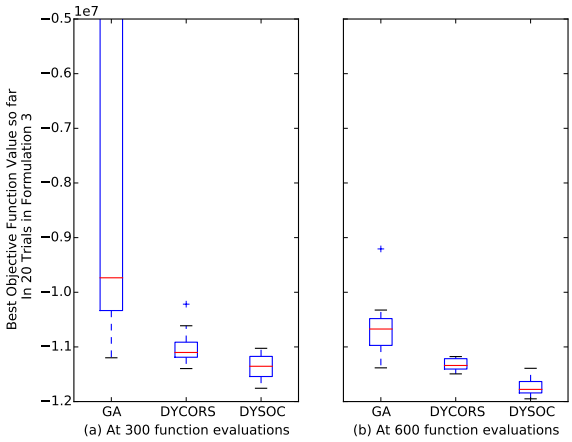


Figure 3.18: Box plot of results among 20 trials in Formulation 3.

Table 3.5: P-values of comparison on pair of different algorithms in 20 trials after 600 function evaluations at 5% significance level. P-value smaller than 5% means that the algorithm in rows is significantly better than the algorithm in column at 95% confidence level (shown as a number with *).

Formulation	Algorithm for	DYCORS	GA
1	DYSOC	$1.61\%^*$	$0.02\%^*$
	DYCORS	1	15.9%
2	DYSOC	$1.55\%^*$	$1.47 \times 10^{-4}\%^*$
	DYCORS	1	$5.92 \times 10^{-2}\%^*$
3	DYSOC	$2.06 \times 10^{-5}\%^*$	$6.30 \times 10^{-6}\%^*$
	DYCORS	1	$1.12 \times 10^{-4}\%^*$

algorithms at 5% significance level, as all p-values are smaller than 5% in three different formulations.

Another way to evaluate the stochastic performance is to check on the reduction of required computational effort for obtaining the same level of results in a statistical way in addition to focus only on the averaged solution. That is to say, we want to find the number of function evaluation N at which we obtain statically indifferent results in algorithm A compared to algorithm B. That is to say after N, the conclusion that 20 data point we get from algorithm A are better than 20

Table 3.6: Number of function evaluations (N) required by DYSOC or DYCORS to reach statistically better results of DYCORS and GA within 600 function evaluations among 20 trials. "NA" means that two algorithms are not significantly different at 5% significance level after 600 function evaluations

Formulation	Benchmark of 600 evaluations	DYSOC	DYCORS
1	DYCORS	336	–
	GA	308	NA
2	DYCORS	449	–
	GA	213	345
3	DYCORS	308	–
	GA	206	249

data points we get from algorithm B is statistically significant. We use a Wilcoxon Rank Sum test for this statistical comparison and significance level is set at 5%.

Table 3.6 shows that in Formulation 1, p-DYCORS is not statistically different compared to p-GA in 600 function evaluations, may be due to the fact that results in p-GA among 20 trials have a large variance as shown standard deviation columns in Table 3.4. But in both Formulation 2 and 3, p-DYCORS only takes around 250$\sim$350 function evaluations to show its significantly better performance than GA. As to p-DYSOC, it needs 449 function evaluations to show significantly better performance than p-DYCORS. And comparing p-DYSOC with p-GA, we need 308 function evaluations Formulation 1, and around 200 in both Formulation 2 and 3. In all, p-DYSOC demonstrate an more efficient performance than both p-DYCORS and p-GA in all formulations and need a small number of function evaluations to show better performance than p-DYCORS and p-GA in statistical analysis among 20 trials.

3.6.3 Evolution of Constraints Values

The change in constrained violation decreases as evalution increases in Figure 3.19, Figure 3.20 and Figure 3.21. By comparing different algorithms, we can see that, p-GA spent relatively more function evaluations in order to find feasible areas, while p-DYCORS and p-DYSOC can focus on searching in feasible solutions fast in general. In Formulation 1, p-DYCORS is the fastest at finding feasible domain, and p-DYSOC catches up at around 200 function evaluations. In Figure 3.19, we can notice that p-DYSOC has the smallest averaged constraint violation values after 200 function evaluations and values are close to zero, meaning that all the trials in p-DYSOC search in least violated domain and the constraints violations contribute the least in objective function compared to p-DYCORS and p-GA. The results indicate that surrogate surfaces built for the constraints enable the algorithm to have a consistently better performance in finding optimal solution.

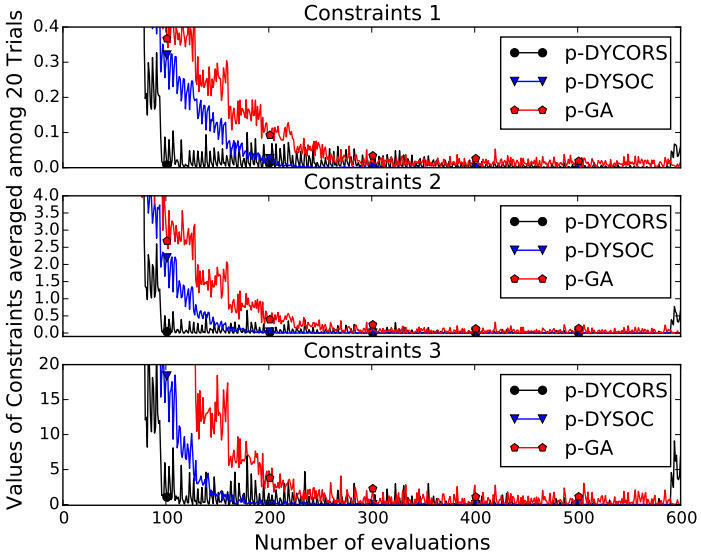


Figure 3.19: Effect of number of iterations vs. the violation of different constraints in Formulation 1. Values are averaged over 20 trials.

Large spikes in p-DYCORS at the end of run in Figure 3.19 are caused by

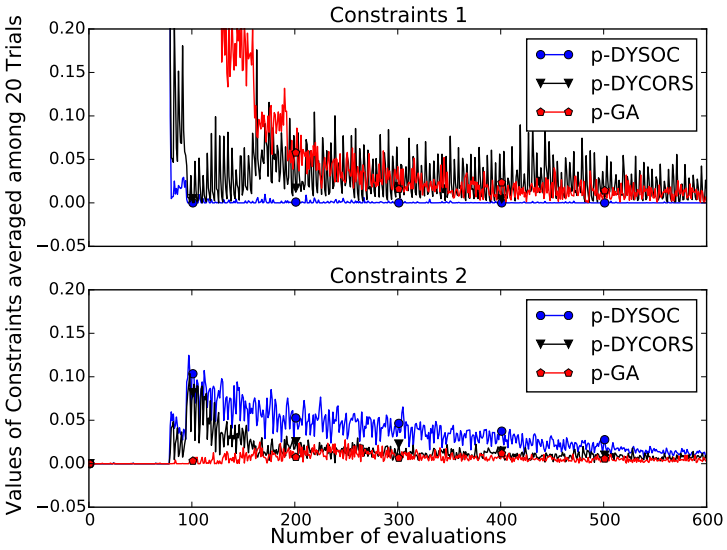


Figure 3.20: Effect of number of iterations vs. the Violation of different constraints in Formulation 2. Values are averaged over 20 trials.

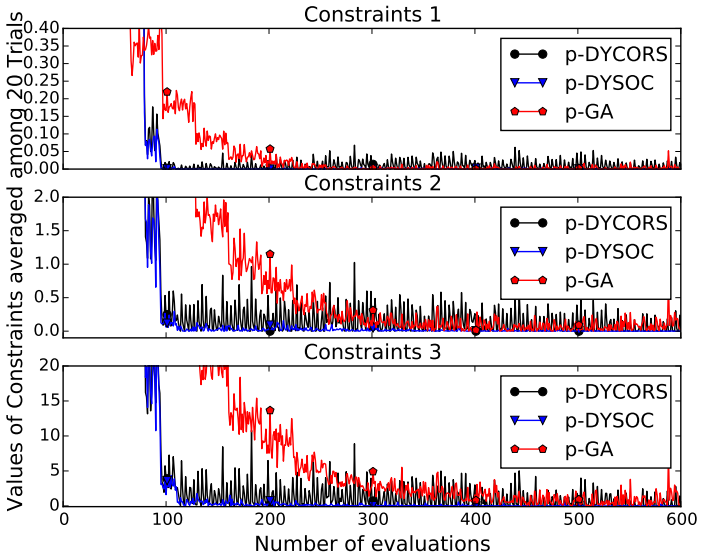


Figure 3.21: Effect of number of iterations vs. the Violation of different constraints in Formulation 3. Values are averaged over 20 trials.

one trial restart, so new random initial design in that trial raises the averaged values in constraint violations, however in p-DYSOC, no restart is triggered in any trial. In Formulation 2, results of the first constraint show that p-DYSOC finds

the feasible area as fast as p-DYCORS while has a consistent performance of all trials. The second constraint in Formulation 2 is easy to satisfy as its averaged random initial design has non violated constraint values (i.e. all zeros), and p-DYSOC unlike p-DYCORS and p-GA keeps searching in infeasible domain with violated solution which indicates that it attempts to find better solution for the objective function. And the performance can be found in the averaged results over trials as in Figure 3.14, showing that p-DYSOC performs the best among all algorithms. Formulation 3 shows that (a) p-DYSOC finds the feasible area as fast as p-DYCORS and (b) p-DYSOC has a consistent performance of all trials as with least fluctuated results in plots for all three constraints. Overall, the surrogate surface built for the constraints help the search focusing on searching on areas where optimal objective function can be found and also reducing the computational effort to find feasible domain.

3.7 Conclusion

Subsidence is a serious issue over large regions of the world. Assessing how best to distribute the groundwater pumping to minimize the occurrence of critical levels of subsidence is an important problem in water resources. It is necessary to have an algorithm that can effectively and accurately solve this problem.

The optimal solutions of Hang-Jia-Hu land subsidence problems obtained have shown an effectiveness of the optimization algorithms. With the help of optimization algorithms, we can resolve the land subsidence problem by redistributing the pumping rates on wells in the region instead of reducing overall groundwater withdrawals or introducing artificial recharge. Moreover, different management

alternatives can be evaluated by using different formulations including increasing groundwater extraction, prioritizing reduction of critical subsidence issues and expanding to extraction from deep aquifer.

The new algorithm used here p-DYSOC combines feature for the earlier algorithm DYCORS [60] and a method for approximating expensive constraint with surrogate. p-DYSOC shows an improved efficiency compared to p-DYCORS and a popular evolutionary algorithm p-GA for Hang-Jia-Hu subsidence problems, which involves both computationally expensive models and large number of decision variables (38) to be optimized. The surrogate surface of objective function helps to reduce the function evaluation required as both p-DYSOC and p-DYCORS outperform p-GA in all three formulations.

In p-DYSOC, the incorporated surrogates of constraints facilitate the surrogate-based optimizer to find the feasible domain and reduce the computation budget significantly in order to obtain a good solution. In both averaged performance analysis and stochastic algorithms analysis, we can find p-DYCORS has the most robust and consistent efficient performance in all the three formulations.

The additional surrogate surface of constraint can help to reduce the computational burden by 50% (as the speed up are over 2 as shown in Table 3.3) in averaged results in all formulations of our test problems. For problems with expensive constraint functions, p-DYSOC shows a promising performance and its application can be extended to different real world problems.

CHAPTER 4

ASYNCHRONOUS-PARALLEL GLOBAL OPTIMIZATION ALGORITHMS WITH EARLY TRUNCATION TECHNIQUE ON COMPUTATIONALLY EXPENSIVE GROUNDWATER CALIBRATIONS

4.1 Introduction

Optimization methods are widely used to estimate model parameters for a variety of water resource problems. The optimization finds the parameter vector x to minimize a loss function $f(x)$ that measures how well the model fits measured data. However, this optimization may be challenging. For many realistic models the evaluations are expensive, so only a few can be made; the underlying simulations are "black-box" codes, so no analytical derivatives are available; and the loss function is a complex, non-convex function of the parameters, which may have many local minima. Especially expensive are simulations of models based on partial derivative equations (PDE) for water flow and contaminant transport (in surface or ground water for example). Surrogate global optimization algorithms (e.g. [39, 28, 5]) are one way to efficiently solve this kind of problem. These methods use a few function evaluations to compute an inexpensive *surrogate model* $s(x)$ that approximates the goodness-of-fit function $f(x)$ and serves to guide further sampling. This surrogate approximation approach to optimization has been shown to reduce the number of costly objective function evaluations necessary to find a good answer ([55, 50]).

In this chapter, we further reduce the time required for calibration of computationally expensive water resource models by *selective early truncation* of evaluations together with an *asynchronous parallel* strategy. For example, in calibrating

a transient water resource model, the loss function might be the sum of squared errors over N successive time steps. Suppose at a proposed parameter value x, the sum of squared errors over the first $K < N$ time steps is much worse than the sum of squared errors over all time steps at the current best solution x_*. Then we need not compute the errors at x from time steps $K+1, \ldots, N$, and can save time by terminating the simulation early. But if we allow early truncation, then different function evaluations may take different amounts of time; as a result, bulk synchronous algorithms that perform batches of simulations together may leave processors idle in each step, as the length of the step is determined by the slowest simulation. In order to avoid this, we employ an *asynchronous* parallel strategy that can start new simulations as soon as processors become available. We illustrate the efficiency of our early truncation algorithm in an asynchronous parallel optimization framework through experiments using the PySOT toolbox for surrogate optimization [17].

4.2 Literature Review

Many previous studies have been dedicated to automatic calibration. Much of this work involves derivative-based approaches such as quasi-Newton [9] and Levenberg-Marquardt methods [11], and these methods have been integrated into software suites such as PEST [13]. But not all third-party codes provide relevant derivatives; and even when derivatives are available, most derivative-based methods are designed to converge to local optimum. Various heuristic optimizers have also been adopted for calibration problems, including genetic algorithms [24, 69], adaptive cluster covering [69], particle swarm optimization and pattern search [29], and differential evolutionary algorithms [29, 10]. Related approaches based on artificial

neural networks [42], rather than posing the calibration problem directly as an optimization, instead fit a meta-model that maps observations to model parameters based on training examples. However, all these methods may require many expensive computational simulations to reach an acceptable solution.

Surrogate-based optimization methods are explicitly designed for complicated and computationally expensive simulation [32, 64]. These algorithms approximate the true, expensive function evaluations by a less computationally expensive approximation (called a surrogate, response surface, or meta-model), and use this approximation to guide further sampling. In high-dimensional spaces, radial basis function interpolants [60] and Gaussian processes (citesimpson2001kriging are often used as surrogates, and a variety of strategies have been proposed to balance *exploitation* of the surrogate to sample near predicted minima and *exploration* of parts of the space where there may be insufficient data for the surrogate to provide accurate predictions.

Many optimization algorithms have been parallelized so that they can make effective use of high-performance computing systems. In many groundwater-related problems such as contaminant source identification, water management, pollution remediation, and model parameterization problems, parallel optimization has been proposed and shown to perform well [43, 48, 61, 22, 73]. But the strategies currently favored in groundwater applications focus on *synchronous* parallelism, i.e. methods in which several function evaluations are started simultaneously, and all must finish before proceeding to the next step. When simulations may have dramatically different run times depending on their inputs, synchronous parallel methods can be very inefficient, as many processors may stay idle waiting for one long-running simulation to end. In this situation, *asynchronous* parallelism may make more

efficient use of HPC systems. Several asynchronous parallel optimization algorithms have been studied, such as asynchronous Tabu Search on multi-commidity application [23], asynchronous evolutionary algorithms [72] in pump scheduling, asynchronous particle swarm [44] on biomechanical problem and the development of asynchronous parallel pattern search [35]. But to our knowledge, none of the previous studies have been dedicated to parameter calibration problems.

In this chapter, we apply an asynchronous parallel surrogate-based optimization algorithm. The algorithm we apply in this chapter is an asynchronous version of the parallel Stochastic Radial Basis Function (SRBF) method [59]. We use a Surrogate Optimization Toolbox (pySOT) [18] which contains options for different surrogate types and parallelism paradigms. We apply the method to a modified real world groundwater calibration model from the Umatilla Super Fund site. The contribution of this chapter is that it is the first use of an asynchronous parallel optimization algorithm with an early truncation strategy for calibration. Truncation threshold is defined in a way to combine both information we obtained in optimization phase and simulation phase. And rules of knowledge extraction are defined to extract the information from the truncated solution and they are incorporated into the asynchronous optimization algorithm. Our results suggest the computational benefits of the approach for other calibration problems with computational expensive models.

4.3 Calibration Problems of Groundwater Model

Calibration of parameters for a PDE model of groundwater flow and transport can be done by combining optimization with relevant data, which includes spatial

distribution of hydraulic heads and concentration observations as well as input data such as pumping rates. Other studies have shown that coupled estimation of flow and transport parameters provides more reasonable and more certain parameter estimates than alternate procedures that use only subsets of observations, e.g., only heads or only concentrations [33].

We work on calibration problems that are based on integrated groundwater flow and transport models. The parameters to be estimated are the hydraulic conductivities distributed on the whole study area. Scenario 1, the "easy site" case, is based on the true distribution of parameters in the model, with nine hydraulic conductivity values, each representing a zone with the same hydraulic conductivity. Scenario 2, the "difficult site" case, has an additional cross-shaped area representing two relatively low permeability zones, each with an individual hydraulic conductivity value. In both cases, we want to obtain the actual distribution of hydraulic conductivities through optimization.

4.3.1 Case Study: Umatilla Superfund Site

Our case study is the calibration problem based on one U.S. Environmental Protection Agency (EPA) groundwater superfund site, Umatilla Chemical Depot (UCD), Oregon, which was contaminated with untreated wash water containing chemical components. Management of this pump and treatment system is one part of a demonstration project, "Application of Flow and Transport Optimization Code to Groundwater Pump and Treat Systems," funded by the U.S. Department of Defense (DoD) and the Environmental Security Technology Certification Program (ESTCP). The detailed DoD/ESTCP study report, model and data can be found on the project website (http://www.frtr.gov/estcp). In our application, a steady

pumping strategy is assumed during a four year management period, and the contaminant TNT is chosen as the only indication chemical to be removed on site, which was a setting based on study from Utah State University contained in the DoD/ESTCP report.

The simulation of the groundwater flow and transport is based on USGS MODFLOW2005 [30] and MT3DMS 5.3 [78] model. The model contains 132 columns, 125 rows and 5 layers, where layer 1 (i.e. the top layer) represents silt and weathered basalt in convertible state between confined and unconfined aquifer and the other layers are confined alluvial aquifers. The hydraulic conductivity in the aquifer is highly heterogeneous especially in layer 1, which varies from 1 ft/day to 5000 ft/day. Zones of different hydraulic conductivity in the MODFLOW model are shown in Figure 4.1. There are ten different hydraulic conductivity zones in the current DoD/ESTCP model; among them one zone has comparatively far lower conductivity than the others as it represents the boundary of an impermeable area. Based on the this configuration, we choose the 9 other hydraulic conductivity zones, which vary from 100 ft/day to 5000 ft/day as our parameters to be calibrated as in Scenario 1. To add more complexity and calibration difficulty in the problem, two less permeable zones are added in Scenario 2, as shown in Figure 4.2.

4.3.2 Observation data

The "observation" data is obtained from MODFLOW and MT3D simulations using the true values of parameters. The goal of the optimization is then to pick the values of the decision vector (e.g. the parameter values) that best fit the synthetic observation data. For Scenarios 1 and 2, the observation data is obtained from the values of configurations given in Figure 1 and 2, respectively.

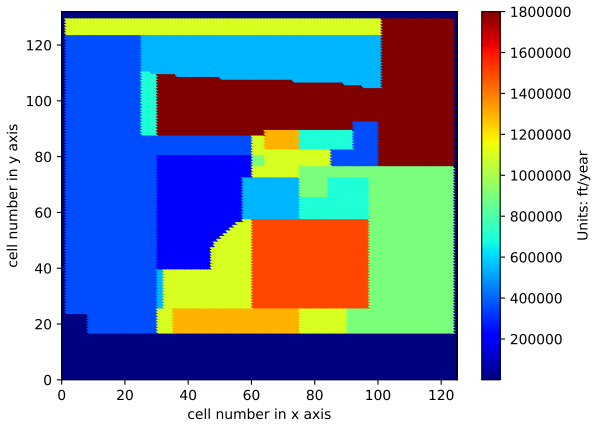


Figure 4.1: Original configuration of Hydraulic Conductivity in Scenario 1

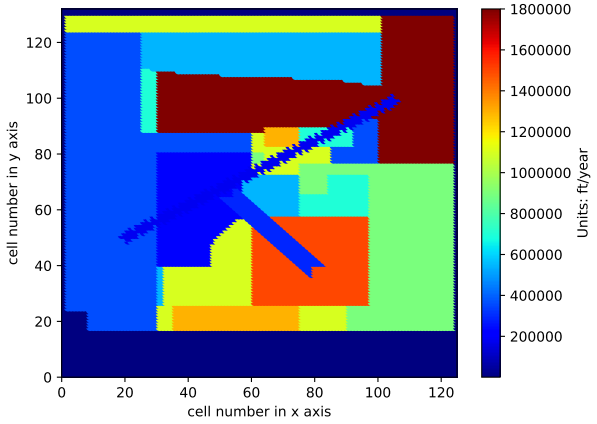


Figure 4.2: Configuration of Hydraulic Conductivity with two additional less permeable areas in Scenario 2

Figure 4.3 illustrates the distribution of all observation wells as well as the hydraulic conductivity zones. These wells are not actual *in situ* wells. We set up the locations of heads monitoring wells randomly in the whole region, while contamination monitoring wells are based on prior knowledge of the migration of the plume. That is to say, wells for contaminations are only designed in the region through which the plume travels in order to better capture the value change. There are in total 107 observation wells for hydraulic heads, and 27 wells for contaminant

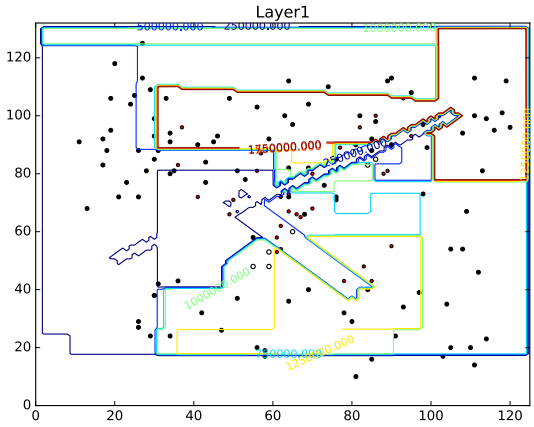


Figure 4.3: Configuration of well locations, with black dots as observation wells of hydraulic head and red dots as observation wells for contaminant concentration, and zones of hydraulic conductivity delineated in colors

concentrations. The observation data for the calibration problem contains one set of hydraulic heads in all 107 wells at the end of the four year management period since pumping rates and heads are constant in time. There is a weekly contamination concentration monitoring data in 27 observation wells. So overall there are 107 observations of heads, and $52 \times 4 \times 27$ observations of contamination concentrations in the four year time period.

4.3.3 Flow and transport model

The MODFLOW flow models for the two scenarios are assumed to be in steady state, with constant hydraulic heads provided for the four year management period, as in the study by Utah State University in the DoD/ESTCP report. In contrast, the MT3D contamination transport model involves an implicit time stepper, and the nonlinear solve required at each transport step may vary in cost depending on the model inputs. Thus, the simulation time required to obtain contaminant

concentrations at a given intermediate time point may vary depending on the model parameters, though the variation is not great. For the models and computing systems used in this study, each MODFLOW simulation requires around $8 - 10$ seconds, while each MT3D simulation of a full management period requires $200 - 250$ seconds.

4.4 Model formulation

4.4.1 Objective function

The goal of calibration is to minimize the deviation between the observation and the simulation. In our problem, the error consists of differences of values in $NH = 107$ head monitoring wells at one time (since the pumping rates are constant); and in $NC = 29$ contamination concentration wells on a weekly basis, as contamination is changing in space and time. The decision variables are hydraulic conductivity values for different spatial zones, which are represented in the decision vector $x \in \mathbb{R}^D$, where $D = 9$ for Scenario 1 and $D = 11$ for Scenario 2. The optimization problem is

$$\min_{x \in \mathbb{R}^D} F(x), \qquad F(x) \equiv G(x, T) \tag{4.1}$$

where T is the total number of transport steps, a constant, and

$$G(x, \tau) = w_1 H(x) + (1 - w_1) C(x, \tau) \text{ , for } 1 \leq \tau \leq T, \tag{4.2}$$

where

$$H(x) = \sum_{n=1}^{N_H} (H_n^{\text{sim}}(x) - H_n^{\text{obs}})^2 \tag{4.3}$$

$$C(x,\tau) = \sum_{m=1}^{N_C} \sum_{t=1}^{\tau} (C_{m,t}^{\text{sim}}(x) - C_{m,t}^{\text{obs}})^2 \text{ for } 1 \le \tau \le T \tag{4.4}$$

subject to the constraints

$$x_i^{min} \le x_i \le x_i^{max}. \tag{4.5}$$

In our model, $x_i^{\min} = 100$ ft/day and $x_i^{\max} = 5000$ ft/day, respectively. H_n^{sim} and H_n^{obs} are the simulated and observed hydraulic head at head observation well n. $C_{m,t}^{\text{sim}}$ and $C_{m,t}^{\text{obs}}$ are simulated and observed contamination concentration at concentration observation well m in week t. Index n in Eqn (4.3) stands for head wells, t stands for time, and m in Eqn (4.4) is for contamination well. T is the total number of transport steps (one each week for four years, i.e. 208). The weight $w_1 = TN_C/(N_H + TN_C)$ compensates for the difference in the number of terms in the sums in Eqn (4.3) and Eqn (4.4).

$G(x,\tau)$ increases monotonically with τ because each term in the sum is non-negative. We consider early truncation of the simulation at a time $\tau < T$ to obtain a lower bound $G(x,\tau) \le F(x)$, where τ is chosen by a truncation strategy described in the next section.

4.4.2 Early truncation strategy

In conventional optimization-based calibration, we would evaluate $G(x,T)$ in Eqn (4.2) for all time steps, so $\tau = T$ in Eqn (4.2). Hence the objective function is "fully evaluated." We can potentially reduce overall computation by prematurely terminating $G(\widetilde{x},\tau)$ for a specific $\widetilde{x}$ and $\tau < T$ if the value of $G(\widetilde{x},\tau)$ in Eqn (4.2)

is poor (e.g. large) compared to value of $G(x,T)$ we have for other x. This is because the full value $G(\widetilde{x},T) \geq G(\widetilde{x},\tau)$ as explained in Section 4.4.1; and, once an unacceptably large sum of errors has been seen in the middle of the simulation when $\tau < T$, there is little benefit to completing the simulation. What we will develop is a method to determine under which condition is stopping the calculation of Eqn (4.4) for $\tau < T$. Thus, while we always let the MODFLOW flow model run to completion, we will develop a method to terminate the more time-consuming transport simulation with MT3D if its error exceeds some threshold. We will call this "early truncation". To clarify how it works, we use Fig. 4.4 to demonstrate a possible time pattern of function evaluations we can obtain in the asynchronous parallelism with early truncation strategy. Assuming we have 12 simulations that are conducted in three cores simultaneously, the total wall clock time required for this task is t_3 which ends at 11^{th} started simulation. Noted here, the numbers on the box of simulation represent the i^{th} started simulation. And two simulations turn out to be early truncated as shown in red boxes. The problem lies in how to determine the threshold for which early truncation makes sense.

4.4.3 Early truncation threshold

Early truncation involves a trade-off between the cost of computation and the benefit of having a computed value of $F(x)$ that can be incorporated into the surrogate. If the computation is terminated after only a few steps, we save computational effort, but learn little about the function. Choosing to terminate later has less benefit, but may be worthwhile if the cost function is much larger than the cost of other points that the algorithm has already explored. At the ith function evaluation, we balance the time savings of early truncation of the ith evaluation

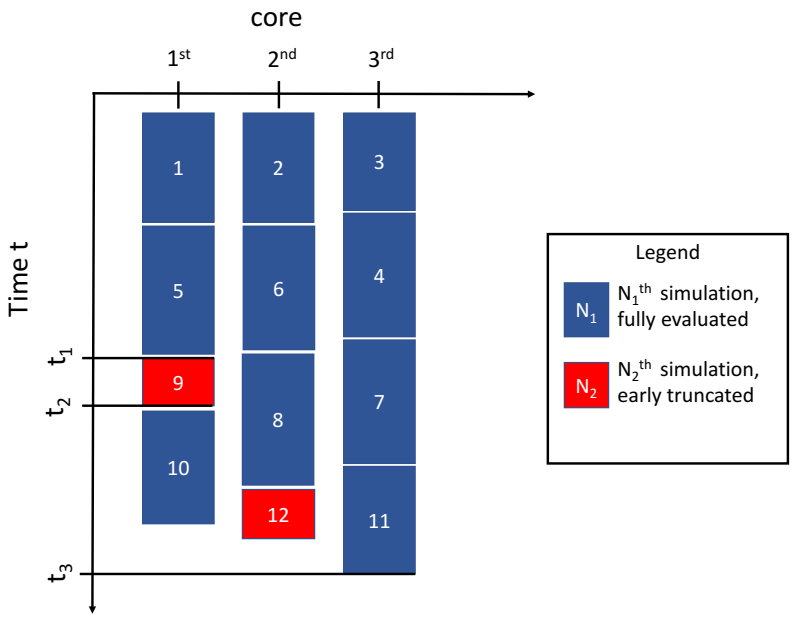


Figure 4.4: Demonstration of time pattern in asynchronous parallelism

against the benefits of completing the evaluation through the criterion

$$\text{truncate if } G(x,\tau) > \Theta(\tau) = L^s(\tau)L_i^p \text{ and } \tau^L \leq \tau \leq \tau^U, \tag{4.6}$$

where L_i^p represents the p^{th} percentile of previously-computed values (or partial evaluations) of the objective function. As shows in Fig. 4.4, the 9^{th} started simulation is truncated, and the component of its truncation threshold is L_6^p as we have 6 simulations finished before the 9^{th} simulation starts. As to 12^{th} started simulation, whether to use L_8^p or L_9^p as part of truncation threshold depends on which rule of knowledge extraction we use as described in Table 4.1 and

$$L^s(\tau) = 1 + \left(\frac{\tau - \tau^U}{\tau^U}\right)^\gamma \tag{4.7}$$

increases monotonically for $\tau \in [\tau^L, \tau^U]$. Truncation does not go into effect until τ^L steps have been taken, nor after τ^U steps have been taken. In our study, we set $\tau^L = 1$ and $\tau^U = 0.9T$.

The functions L_i^p and $L^s(\tau)$ are used to adapt to the stage of the optimization

algorithm and the work spent on each function evaluation, respectively. The value of L_i^p decreases as the optimization proceeds and more values are computed, so the algorithm can be more aggressive about early truncation later in the optimization process. We choose $p = 75$ in this study. The idea with $L^s(\tau)$ is that since $G(x, \tau)$ is a sum of positive terms, we should terminate earlier for smaller values of τ. Various linear or nonlinear equations for L^s may be used; in our study, we choose a cubic polynomial ($\gamma = 3$ in Eqn 4.7).

Figure 4.5 illustrates the role of early truncation. In the figure, the top curve L_{1000}^p shows the shape of threshold function $\Theta(\tau)$ at iteration 1000. If we assume a simulation has objective function $G(\widetilde{x}, \tau)$ based on the vector of parameters as $\widetilde{x}$, and value of $G(\widetilde{x}, \tau)$ is given by the green curve in Figure 4.5, then the evaluation of $G(\widetilde{x}, \tau)$ would stop at black cross with τ^*, at which $G(\widetilde{x}, \tau)$ exceeds the threshold $\Theta(\tau)$.

4.5 Optimization algorithm

4.5.1 SO-SP

[57] describe a strategy for finding the global optimum of a computationally expensive function using a *metric response surface* (MRS). The idea is that in each iteration of the algorithms, an approximation of the objective function $F(x)$ (also called a response surface or surrogate) is constructed based on all the values $(x, F(x))$ computed in previous iterations. In our study, we use an MRS strategy with synchronous parallelism called SO-SP (Surrogate optimization with synchronous parallelism). SO-SP combines two algorithms, parallel LMSRBF ([57])

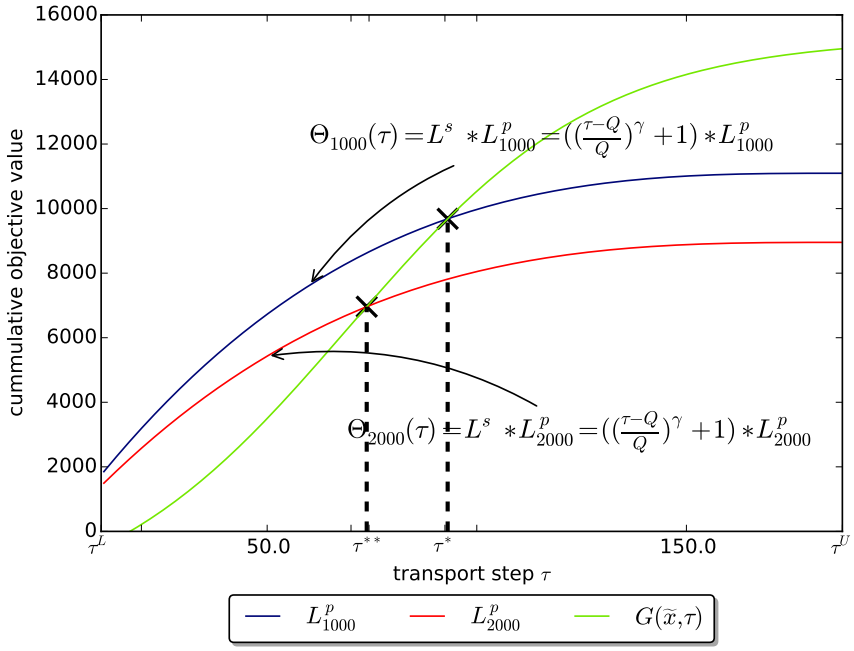


Figure 4.5: Early Truncation Strategy with an example of simulation $G(\widetilde{x}, \tau)$, where two Early truncation thresholds are shown as $\Theta_i(\tau)$ for $i = 1000$ and $i = 2000$. Here i represents iteration i, $G(\widetilde{x}, \tau)$ is the objective function as an SSE of simulation results, which would be truncated at τ^* when $i = 1000$ or at τ^{**} at $i = 2000$

and DYCORS ([60]), and it is implemented in our open source software pySOT (https://github.com/dme65/pySOT).

The detailed steps in SO-SP are described in Algorithm 2. In steps 1 to 3 of Algorithm 2, we use an experimental design (i.e. a Latin hyper cube) to create the initial surrogate approximation. Rather than fitting the objective function $F(x)$, we fit the *capped* function

$$F_{capped}(x) = \min(F(x), \text{median}_{n \in \mathcal{A}_i} F(x_n)) \tag{4.8}$$

where $\mathcal{A}_i$ is the set of indices of points evaluated at round i. By capping the function, we seek to avoid oscillations in the surrogate associated with very large function values (and gradients) in regions far from the minimum.

We then update parameters in algorithms and check on whether we need to restart the whole algorithm or not by using function **Restart_Criterion_Check**. In **Restart_Criterion_Check**, we keep track of the number of consecutive success (i.e. $\mathcal{C}_{success}$) and number of consecutive failure (i.e. $\mathcal{C}_{success}$) in order to determine the search radius σ_i at iteration i. The criterion of "success" iteration is that the best point we found in iteration i (i.e. x_i^*) is better compared to all the points we evaluated so far (i.e. $F(x_i^*) < f_{best}$), otherwise, this iteration is a failure. Thresholds $\mathcal{T}_{success}$ and $\mathcal{T}_{fail}$ are pre-defined values. And once the number of reduction of search radius reaches to the pre-defined r_{max}, we will restart the algorithm from scratch. The restart technique helps the search to escape from a local optimum.

From Step 6, we start to determine the points for function evaluation by first generating Candidate Points set Ω_i, which is generated from using a normally distributed perturbation amplitude around best points evaluated so far (i.e. x_{best}), with a declining probability of perturbation occurring for each dimension as described in Eqn (4.9).

$$p(i) = p_0[1 - \frac{log(i - i_{init} + 1)}{log(i_{max} - i_{init})}] \tag{4.9}$$

Where i is the current number of iterations. i_{init} is the number of iterations for initial design. i_{max} is the total number of iterations, and p_0 is the initial probability. Therefore, $p(i)$ is a decreasing function of i. The procedure for candidate points was suggested in [60] and Eqn (4.9) was suggested by Tolson and Shoemaker [71] for the Dynamic Dimensional Search (DDS) algorithm. **Selection_Evaluation_Points** function described in the function box is used to select points to be included in set of evaluation points (i.e. $\mathcal{D}_i$).

There are two metrics considered in selecting the point for $\mathcal{D}_i$: (1) the estimated values evaluated on the current surrogate surface; (2) the distance infor-

mation from previously evaluated points to the new one. The basic concept is to choose the point which has a good (i.e. low) estimated value and also has large distance to the closet previously evaluated points including selected points for expenin order to explore more in the area that has not been searched. That is to say, as we need to generate P number of evaluation points (i.e. $size(\mathcal{D}_i) = P$), **Selection_Evaluation_Points** function need to be processed for P times, one for each point generation. In Step 7, SO-SP start all cores (i.e. P cores) at the same time for evaluating P points in $\mathcal{D}_i$ for expensive functions results. After all computational resources (i.e. P cores) finish evaluations, we can then update both the set of evaluated points $\mathcal{A}_i$ as well as the response surface based on set $\mathcal{A}_i$. This iteration loop forms the basic structure of SO-SP algorithm.

We implement SO-SP by using Surrogate Optimization Toolbox (pySOT in [17]). The realization of the parallel paradigm in pySOT is based on POAP, which is an event-driven framework for building and combining optimization strategies. There are two parts in POAP, controller and strategy. The controller communicates between worker cores and master core in order to distribute evaluation jobs. The master core which runs the strategy is responsible of building surrogate surface and selecting Candidate Points for evaluation. In this study, we use MPI (message passing interface) version of POAP for the parallel communication in High Performance Computing system, and as to strategy, we use RBF surrogate surface and CandidateDYCORS method for generating the evaluation points as discussed in Eqn (4.9) which has an additional defined probability as a decreasing function against iteration i to select certain coordinates in decision variables for perturbation in order to obtain set Ω_i.

Algorithm 2: SO-SP (Synchronous Parallel)

Input: Initial experimental design, Sample point strategy, Surrogate model, Stopping criterion, Restart criterion

Output: Best solution and its corresponding function value

1: Generate an initial experimental design $\mathcal{I}$;

2: Evaluate the points in the experimental design by F(x) , and initiate the set of evaluated points (i.e.$\mathcal{A}_0 := \mathcal{I}$);

3: Build a Surrogate RBF model $s_0(x)$ for points in $\mathcal{B}_0 = \{(x, F_{capped}(x)) : x \in \mathcal{A}_0\}$;

repeat

  if Restart_Criterion_Check then

    4: Reset the Surrogate RBF model and the Sample point strategy;

    5: go to (1) ;

  end

  6: Generate Candidate Points set Ω_i by a normally distributed perturbation amplitude around best points evaluated so far (i.e. x_{best}), with a declining probability of perturbation for each dimension as in Eqn (4.9). Here $x_{best} = \underset{x}{\operatorname{argmin}} F(x)$ for $x \in \mathcal{A}_{i-1}$;

  for $p \in \{1, 2, ..P\}$ **do**

    7: Select x_i^p evaluation point among Candidate Point Ω_i and included it in set $\mathcal{D}_i$ which is the set of points to be evaluated in parallel. The selection is based on metrics function $s_{i-1}(x)$ i.e. **Selection_Evaluation_Points** $(\Omega_i, \mathcal{B}_i, s_{i-1}(x))$;

  end

  8: Evaluate f(x) for all x in $\mathcal{D}_i$ generated using P cores simultaneously;

  9: Update the set of evaluated points as $\mathcal{A}_i = \mathcal{A}_{i-1} \bigcup \mathcal{D}_i$ and the surrogate RBF model $s_i(x)$ based on values in set $\mathcal{B}_i$ for $\mathcal{B}_i = \{(x, F_{capped}(x)) : x \in \mathcal{A}_i = \mathcal{A}_{i-1} \bigcup \mathcal{D}_i\}$;

until *Stopping criterion met*;

4.5.2 SO-AET

SO-AET is the new algorithm we introduce, which combines features from SO-SP and asynchronous parallel and it incorporates a new "early truncation function" as discussed in Section 4.4.2 and Table 4.1. The diagram of this SO-AET can be found in Figure 4.6.

function $x_{i+1} =$ **Select_Evaluation_Point**$(\Omega_i, \mathcal{B}_i, s_i(x))$
(a) *(Estimate Function Value of Candidate Points)* For each $x \in \Omega_i$, compute the RBF value $s_i(x)$;
(b) *(Determine Minimum Distance from Previously Evaluated Points)* For each $x \in \Omega_i$, compute $\Delta_i(x) = \min_{1 \le n \le i} \|x - x_n\|$. (Here,$\|\cdot\|$ is the Euclidean norm on $\mathbb{R}^D$) ;
(c)*(Compute Weighted Score and Select Next Evaluation Point)* For each $x \in \Omega_i$, compute $\mathcal{W}_n(x) = \theta_1(s_i(x) - s_i^{min}) + \theta_2(\Delta_i(x) - \Delta_i^{min})$, and find x_{i+1} as the point in Ω_i that minimizes $\mathcal{W}_i$.
More details of θ_1 and θ_2 (which include normalization functions) can be found in [57].

function $Flag =$ **Restart_Criterion_Check**$(x_i^*, params)$
where $params = (\sigma_{i-1}, \mathcal{T}_{fail}, \mathcal{T}_{success}, \mathcal{C}_{fail}, \mathcal{C}_{success}, r_{fail}, r_{max})$
(a) *(Update counter)* **if** $F(x_i^*) < f_{best}$ **then** reset $\mathcal{C}_{success} := \mathcal{C}_{success} + 1$ and $\mathcal{C}_{fail} := 0$;
else reset $\mathcal{C}_{fail} := \mathcal{C}_{fail} + 1$ and $\mathcal{C}_{success} := 0$;
(b) *(Adjust step size)* **if** $\mathcal{C}_{fail} \ge \mathcal{T}_{fail}$ **then** $\sigma_i = max(\sigma_{i-1}/2, \sigma_{min})$, $\mathcal{C}_{fail} = 0$ and $r_{fail} = r_{fail} + 1$ **else if** $\mathcal{C}_{success} \ge \mathcal{T}_{success}$ **then** $\sigma_i = 2\sigma_{i-1}$, and $\mathcal{C}_{success} = 0$;
(c)*(Check on restart)* **if** $r_{fail} > r_{max}$ **then** Flag = True **else** Flag = False

Algorithm 3 describes the major steps in SO-AET. As a start, we evaluated points in the initial experimental design by objective function $F(x)$ and initialize set of Distance Relevant Points as $\mathcal{A}_0^e := \mathcal{I}$ and set of Surface Relevant Points $\mathcal{A}_0^s := \mathcal{I}$ in Step 2. Definition of both "Distance Relevant" and "Surface Relevant" are described in Table 4.1. Both Distant relevant points and Surface relevant points are used in function **Selection_Evaluation_Points** to select evaluation points. And Surface relevant points are also used in construction of an update surrogate approximation $s_n(x)$ in each iteration in Algorithm 3 Step 12. Then Initial surrogate surface $s_0(x)$ was built by capped objective function as in Eqn (4.8) in Step 3. Then, we use **Restart_Criterion_Check** to determine whether to restart or not similar as in SO-SP. If no restart is needed, in Step 6, in order to obtain function evaluation i, we generate Candidate Points Ω_i by perturbing

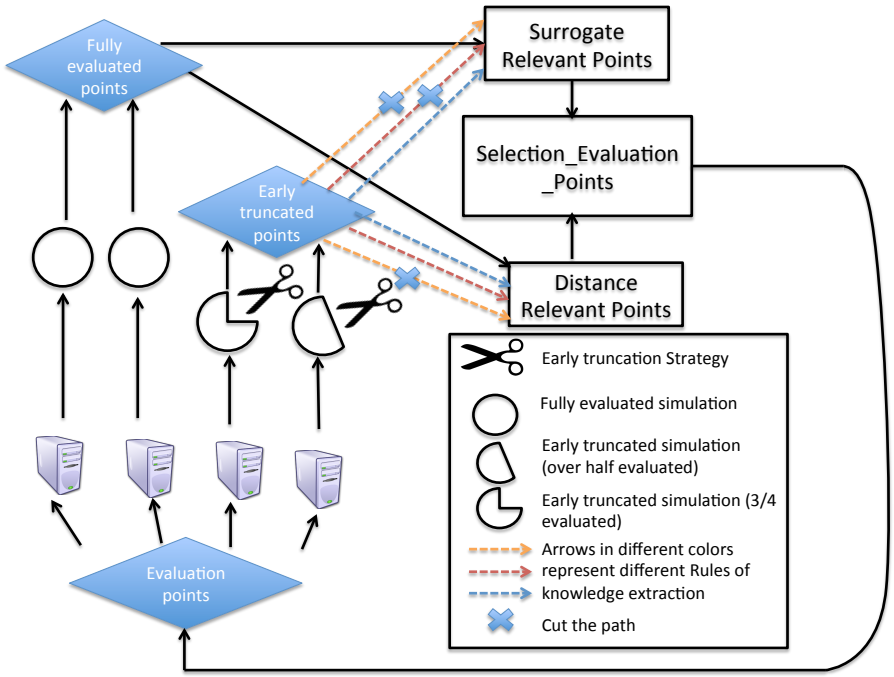


Figure 4.6: Framework of SO-AET-k. For dotted arrow part, orange dotted arrows represent SO-AET-1 with both orange paths cut based on the rule k=1 that means it does not provide any information of truncated points, red arrows represent SO-AET-2 which does not provide information of truncated points to update surrogate surface based on the rule k=2, blue arrows represent SO-AET-3 which retain both distance and values information of truncated points

selected decision variables around the best solution found so far x_{best}.

In Step 7, the evaluation point x_i is selected based on metrics function **Select_Evaluation_Point** among Candidate Points one at a time, and then it is provided to the next available computational core for expensive function evaluation. The point x_i is evaluated by simulation model with the early truncation strategy described in Eqn (4.6) in Step 8. As in asynchronous paradigm, different cores can carry out Step 8 at different time. Each time a core finishes the evaluation of x_i, there are two outcomes due to Eqn (4.6): (1) x_i is an early truncated point. (2) x_i is a fully evaluated point. For fully evaluated point, we should follow the same updating technique in SO-SP. That is to say, x_i is categorized as both

Distance Relevant Point (i.e. x_i^e) and Surface Relevant Point (i.e. x_i^s) in Step 9 in Algorithm 3. If x_i is a truncated point, we record the value of x_i and predict its "full objective value" L_p as in Eqn (4.10). And then we can determine the category of x_i (i.e. Distance Relevant Point or Surface Relevant Point or both) based on three different Rules of Knowledge Extraction (SO-AET-k for $k \in \{1, 2, 3\}$) which are proposed to help us determine how to incorporate the information of x_i. The definition of the three different rules of knowledge extraction are also shown in Table 4.1. The value of Eqn (4.10) is used to build response surface for the points either truncated or not based on different SO-AET-k. So that we can finish updating set of Distance Relevant Points and set of Surface Relevant Points in Step 11 and Step 12.

$$y(x) = \begin{cases} w_1 * H(x) + (1 - w_1) * C(x) & \text{, if not truncated} \\ L^p & \text{, if truncated} \end{cases} \tag{4.10}$$

Algorithm 3: SO-AET (Asynchronous Parallel with Early Truncation)

Input: Initial experimental design, Sample point strategy, Surrogate model, Stopping criterion, Restart criterion, Rules of Knowledge Extraction (SO-AET-k, k $\in$ {1,2,3})

Output: Best solution and its corresponding function value

1: Generate an initial experimental design $\mathcal{I}$;

2: Evaluate the points in the experimental design by F(x), and form the set of distance relevant points (i.e.$\mathcal{A}_0^e := \mathcal{I}$) and the set of surface relevant points (i.e.$\mathcal{A}_0^s := \mathcal{I}$), definitions of distance relevant and surface relevant points are listed in Table 4.1;

3: Use Eqn (4.8) to compute $F_{capped}(x)$ and build a Surrogate RBF model $s_0(x)$ based on points in $\mathcal{B}_0 = \{(x, F_{capped}(x)) : x \in \mathcal{A}_0^s\}$;

Continue on next page

The explanations of the difference in synchronous and asynchronous parallelism are as follows. In synchronous parallel version, if any core finishes its job early, it needs to wait for all the other cores to finish in order to proceed on. In each iteration, with P cores used, we update the RBF surface based on results of P

Table 4.1: Definition of three different Rules of Knowledge Extractions (SO-AET-k for k $\in \{1, 2, 3\}$) and categories of x_i

Definition
Knowledge Extraction 1 (SO-AET-1) : This is the setting in which we do not keep any information of x_i if x_i is determined as an early truncated point, that is to say, x_i is regarded as the point we have never evaluated so far. In Algorithm SO-AET Step 10, x_i is regarded neither as x_i^e nor as x_i^s. Therefore, $\mathcal{A}_i^e = \mathcal{A}_{i-1}^e$ in Step 11 and $\mathcal{A}_i^s = \mathcal{A}_{i-1}^s$ in Step 12. And the component in truncation threshold, L_i^p as the p^{th} percentile of all the evaluated points but excluding x_i.
Knowledge Extraction 2 (SO-AET-2) : This is the setting that we only keep distance information if x_i is an early truncated point, its evaluated value is not used for building the response surface. Therefore, in Algorithm SO-AET, x_i is regarded as x_i^e but not x_i^s in Step 10. And $\mathcal{A}_i^e = \mathcal{A}_{i-1}^e$ in Step 11, but $\mathcal{A}_i^e = \mathcal{A}_{i-1}^s \bigcup x_i^s$ in Step 12. Here the component in truncation threshold, L_i^p is defined as the p^{th} percentile of all the evaluated points excluding x_i.
Knowledge Extraction 3 (SO-AET-3) : This is when we keep both the distance information and the "predicted" evaluation results for x_i. That is to say, when building the response surface, we use an estimation of objective values for x_i. The idea behind this approach is that as we do not waste any information obtained from the early truncated points. Therefore, x_i is regarded as both x_i^s and x_i^e. And both $\mathcal{A}_i^e$ and $\mathcal{A}_i^s$ get updated as in $\mathcal{A}_i^e = \mathcal{A}_{i-1}^e \bigcup x_i^e$ and $\mathcal{A}_i^s = \mathcal{A}_{i-1}^s \bigcup x_i^s$. For building the response surface, we take those early truncated points into consideration in the way as follows, we use the threshold values L^p as an "estimation" of their fully evaluated objective results. So our actual objective values is computed in the form as shown in Eqn (4.10). And to L_i^p, we define it as the p^{th} percentile of all the evaluated points including x_i.
Surrogate Relevant x_i (x_i^s) : x_i is surrogate relevant for SO-AET-3 since its value of $f(x_i)$ is used to build the surrogate under rule of SO-AET-3 or if x_i is fully evaluated.
Distance Relevant x_i (x_i^e) : x_i is distance relevant for both SO-AET-2 and SO-AET-3 cases because the value of x_i is used in the distance calculation for weighting candidate points under rule of SO-AET-2 or SO-AET-3 or if x_i is fully evaluated.

Algorithm 3: (Continued) SO-AET (Asynchronous Parallel with Early Truncation)

repeat

  if x_{i-1} *is fully evaluated point* **then**

    if Restart_Criterion_Check then

      4: Reset the Surrogate RBF model and the Sample point strategy;

      5: go to (1);

    end

  end

  6: Generate Candidate Points set Ω_i based on normally distributed perturbation amplitude around best points evaluated so far, with a declining probability of perturbation occurring for each dimension i.e. $x_{best} = \underset{x}{\operatorname{argmin}} F(x)$ for $x \in \mathcal{A}^e_{i-1}$;

  7: Select evaluation point x_i to be evaluated among Candidate Points Ω_i based on metrics function $s_{i-1}(x)$ i.e. **Selection_Evaluation_Points** $(\Omega_i, \mathcal{B}_i, s_i(x))$;

  8: Evaluate f(x) at the point x_i with possible early truncation determined by Eqn (4.6) and (4.7);

  if x_i *is fully evaluated point* **then**

    9: x_i is both a surrogate relevant point (i.e. x^s_i) and also distance relevant point (i.e. x^e_i);

  else

    10: Based on rules of Knowledge Extraction (SO-AET-k) as defined in Table 4.1, determine whether x_i is a Surrogate Relevant point (i.e. x^s_i) and also whether x_i is Distance Relevant point (i.e. x^e_i);

  11: Update the set of distant relevant points $\mathcal{A}^e_i$ as $\mathcal{A}^e_i = \mathcal{A}^e_{i-1} \bigcup x^e_i$ based on availability of x^e_i;

  12: Update the surrogate RBF model $s_i(x)$ based on surface relevant values in set of $\mathcal{B}_i$ for $\mathcal{B}_i = \{(x, F_{capped}(x)) : x \in \mathcal{A}^s_i\}$ where $\mathcal{A}^s_i = \mathcal{A}^s_{i-1} \bigcup x^s_i$ is updated if x^s_i is available;

until *Stopping criterion met*;

evaluations in each iteration, set of informative evaluated points and internal parameters in SO-SP are updated once every iteration.

By contrast, for SO-AET with asynchronous parallelism, each core can start its evaluations once it finishes its job without considering the working status of other cores. Consequently, surrogate surface, informative evaluated points as well as internal parameters in the algorithm are updated soon after each function evaluation instead of being waiting for other cores to finish. Therefore, asynchronous parallelism prevents cores from being idle in the optimization processes. For computationally expensive functions that take various times given different inputs, the overall computational budget for finding the optimum inputs can be reduced with asynchronous parallel. If there is no variability in objective function computation time, then the synchronous parallel algorithms parallel algorithms SO-SP is possibly better. With synchronous optimization, each new response surface is based on all P evaluation of the objective function (with P cores). If there are differences (even small ones) in computation time, then the surrogate will be updated after each objective function evaluation using asynchronous parallel, so the algorithm results can be different and we expect the advantages of asynchronous to increase as the variability of the objective function evaluation time increases.

4.5.3 Other algorithms

We will compare SO-AET and SO-SP to other algorithms. There are no codes available for other global optimization algorithms that are also available in asynchronous parallel, so the comparison are one asynchronous local optimizer and one synchronous global optimizer.

4.5.3.1 APPSPACK

Asynchronous Parallel Pattern Search Pack (APPSPACK) toolbox solves for non-linear optimization problems. It is a local optimizer, but does not require derivative information. The toolbox is based on Asynchronous parallel pattern search (APPS) algorithm [35]. In APPS, trial points are iteratively generated according to the parameters defined as search direction and step size. Based on whether the best evaluation of trial points is found in search or not, the search step is identified as successful or unsuccessful step; and in consequence the direction and step size change in different fashion. Successful step changes direction of the search, whereas unsuccessful step narrows the selection of trials points around the current best solution. As iteration goes on, the search domain would be narrowed to the optimal point. But since it is local optimization method, the search may be trapped in a local optimum. Another characteristic of APPS is that it is designed for asynchronous parallelism, so that there is no idle time. The implementation of this asynchronous parallelism is based on MPI in C++ version, and we integrate it with the python version of Umatilla calibration problems, that is the objective function. In addition, we couple the early truncation technique with APPSPACK to introduce as a new strategy APPSPACK-AET and test its performance on the Umatilla calibration problems for algorithm comparison.

4.5.3.2 SCE-UA

Shuffle Complex Evolutionary Algorithm (SCE-UA) [16, 15] is a general-purpose global optimization method although it has been used primarily for calibrating water resource models. It also does not require any derivative information. The basic concept of this algorithm is to systematically evolve complexes of points span-

ning in the decision domain in search of the global optimal. During the evolution, competitive selection and complex shuffling are used to ensure search direction for better objective value, and the combination of deterministic and probabilistic ways of searching makes SCE-UA flexible and robust. SCE-UA has been widely used in water resources research.

In this study, we use a python version of SCE-UA in SPOTPY [36] which is a Statistical Parameter Optimization Framework for Python.

4.6 Results and Discussion

4.6.1 Truncation strategy analysis

The truncation policy is implemented to reduce computation time in objective function evaluation.The truncation pattern examined numerically varies for different trials and different truncation policies. In order to understand the performance of the truncation setting, we plot truncation patterns of three different Rules of Knowledge Extraction (SO-AET-k) (Table 4.1) and evaluation results for one trial in Scenario 1 as an example of the impact of the truncation policy. In addition we compute the proportion of truncation points (Φ) at the end of 10000 seconds as in Eqn (4.11) for different rules (i.e. $k \in \{1, 2, 3\}$) for two scenarios.

$$\Phi(i,k) = \frac{\text{number of truncated points from iteration 0 to i based on rule } k}{\text{number of all evaluated points from iteration 0 to i}} \tag{4.11}$$

For a good truncation strategy, on the one hand we need to truncate many points to allow certain variations in the computation time of simulations in order

to benefits from the asynchronous paradigm. This then creates a variability in the time required to compute each objective function. What we can see from Figure 4.7, results in the first trial in Scenario 1 show that simulation time varies significantly and spans over the range of 300 seconds with the truncation strategy. On the other hand, we cannot truncate too many points, because we need an adequate number of points to provide enough information to build accurate response surface. A poor result would be that we can have few non-truncated point in optimization because truncation level is too harsh. As an example, Figure 4.7 shows that we don't have this type of problem with the truncation threshold we defined in Eqn (4.6) and (4.7) since there are many non-truncated points.

Table 4.2 describes the mean value (i.e. μ) and standard deviation (i.e. σ) for the best result obtained (i.e. *best_eval*) among 30 trials and the proportion of truncation (i.e. Φ) as described in Eqn (4.11). It shows that for both scenarios the best average and lowest variance are obtained by SO-AET-3 strategy. The Knowledge Extraction strategy in SO-AET-3 uses the truncated points both in the distance calculation for candidate points selection (Algorithm 3 Step 7 or **Selection_Evaluation_Points**) and the value of the objective function when the truncated points are used in building the surrogate S_i.

As we discussed in Section 4.4.3, L_i^p is used to denote the part of threshold level during the optimization phase. Figure 4 shows that L_i^p is decreasing as i increases based on numerical results computed for Scenario 1 for $i = 1000$ and $i = 2000$. This trend indicates that with more iterations of the optimization, more good solutions are being evaluated, so the p^{th} percentile value is lower. Since L_i^p gets smaller with increasing iteration i, the truncation part of the algorithm becomes more aggressive, i.e. the truncation threshold gets lower. It is sensible

Table 4.2: Statistics of Proportion of Truncation (including mean $\mu_{\Phi(10000,k)}$ and standard deviation $\sigma_{\Phi(10000,k)}$) and Best Value (including mean μ_{best_eval} and standard deviation σ_{best_eval}) among 30 trials by using different Rules of Knowledge Extraction (i.e. SO-AET-k for k=1,2,3). All cases are run for 10000 seconds. Best solutions are bolded.

Scenario	setting	μ_{best_eval}	σ_{best_eval}	$\mu_{\Phi(10000,k)}$	$\sigma_{\Phi(10000,k)}$
1	SO-AET-1	10.391	10.234	79.2%	12.2%
	SO-AET-2	6.958	3.674	12.9%	3.5%
	SO-AET-3	**6.270**	1.856	8.3%	2.0%
2	SO-AET-1	11.001	22.184	29.0%	5.3%
	SO-AET-2	14.096	25.945	21.5%	6.0%
	SO-AET-3	**6.223**	16.556	13.0%	1.8%

that the algorithm would become more aggressive as it obtains more information to facilitate searching the optimum.

4.6.2 Progress graph analysis

To analyze the solution of synchronous and asynchronous paradigm, we ran each setting (SO-SP and SO-AET-k for $k \in \{1, 2, 3\}$) for 30 trials using 16 cores and measured their results against the wall clock time used. All settings start with the same 30 sets of initial designs, one for each trial.

The progress graphs we use in this section describe the solutions of the averaged best results obtained over 30 trials against the wall clock spent. Figure 4.8 and Figure 4.9 are progress graphs for two scenarios. In both figures, we can see that SO-AET-k with different Rules of Knowledge Extraction show dominantly superior performance compared to SO-SP, as their related curves all stay lower than their synchronous version at any time. That means the averaged performance of SO-AET-k for all k can obtain solutions with good objective values sooner

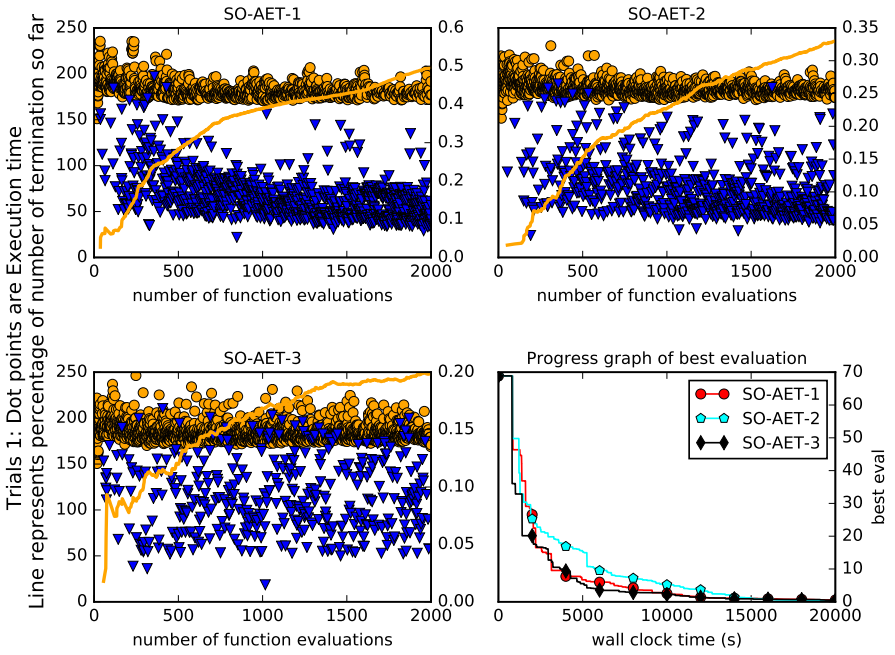


Figure 4.7: Results of truncation in terms of both proportion of number of truncated points and the progress plot on best result found so far by using different rules of Knowledge Extraction (i.e. k=1 or k=2 or k=3) combined with SO-AET in one trial in Scenario 1, where three plots with yellow dots and blue dots describe the execution time for each evaluations, blue dots are early truncated evaluations and yellows dots are fully executed evaluations, the yellow curves on the three plots represent the evolution of $\Phi(i, k)$ along number of function evaluation i. The forth plot "Progress graph" illustrates the evolution of the best result find so far comparing against three different settings (i.e. SO-AET-k for k=$\{1, 2, 3\}$) in that trial in Scenario 1.

than synchronous paradigm. In comparison among different Rules of Knowledge extraction (k), results from Scenario 1 show that SO-AET-2 obtains averaged results slightly worse than the other two rules at the end of 10000 seconds, whereas in Scenario 2 averaged curve of SO-AET-1 shows a slight worse performance than the other two rules. Overall, SO-AET-3 performs the best in both scenarios. This indicates that even though the values reserved from truncated points are estimation of full evaluation, they can be valuable in building the response surface, as shown in our test experiments. However, theoretically, cases exist when early truncated

points are falsely estimated as unpromising solution, while in reality, they can yield good outputs. In this calibration problem, the error sum along simulation transport time might not have such problems, as the discrepancies in parameters should result in relatively similar amount of errors in observations at different time steps.

In addition, we compare performance of SO-SP and SO-AET-k with two other algorithms, i.e. SCE-UA, APPSPACK and with APPSPACK-ET is APPSPACK plus our early truncation strategy and Knowledge Extraction Rule ($k = 3$). The truncation threshold of APPSPACK-ET is the same as the one we use for SO-AET-k as described in Section 4.4.3.

To ensure a fair comparison, we use the best point found in initial design of SO-SP and SO-AET-k as the starting point in APPSPACK and APPSPACK-ET, and also for each trial of SCE-UA. In the progress graphs of averaged performance (Figure 4.8 and Figure 4.9) for 2 scenarios, the averaged results of SCE-UA are worst along the optimization and the second largest results are from APPSPACK-ET and APPSPACK ranks the third worse. The early truncation strategy does not improve the performance of APPSPACK as averaged results from APPSPACK-ET are worse than APPSPACK along the optimization in both scenarios. We can see the statistical comparison of APPSPACK and APPSPACK-ET in box plot in Section 4.6.3.

To quantify the efficiency of different algorithms, we check on the wall clock time required to obtain a level relatively hard to achieve, but same for all different algorithms. In most cases, after many iterations in optimization, even a small amount of improvement in result would require a long time of optimization, therefore if an algorithm variant can reach a better objective value much earlier than the

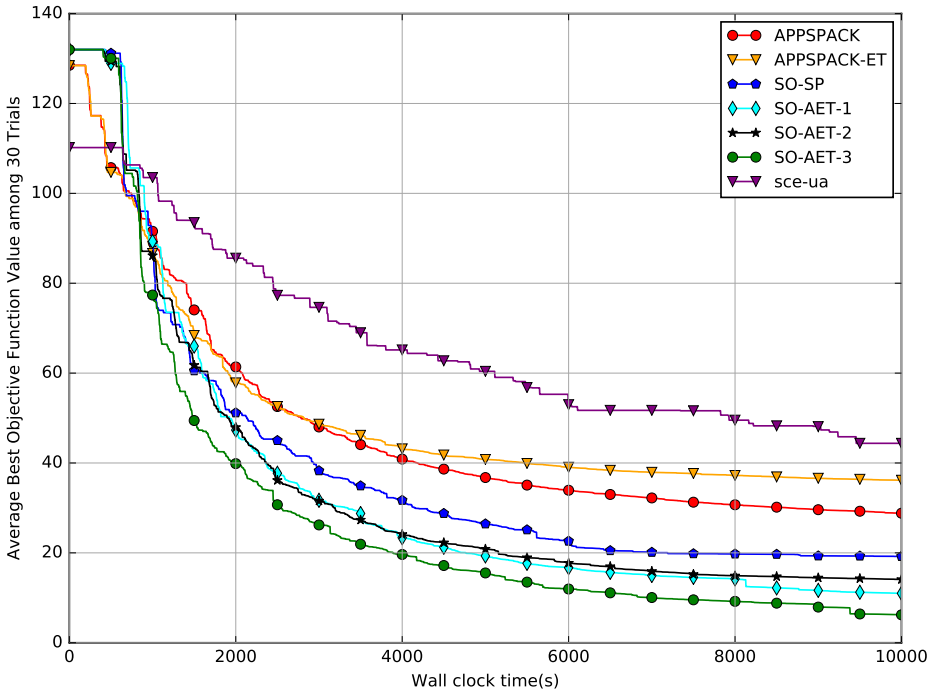


Figure 4.8: Progress Graph of results Scenario 1 which compare averaged best results among 30 trials found so far at each wall-clock time for different Rules knowledge extraction of SO-AET-k with its synchronous version SO-SP, and other two algorithms as SCE-UA, APPSPACK and APPSPACK-ET (i.e. APPSPACK with Early truncation defined same as in Section 4.4.3)

other is of great benefit. As a result, we compared each of the asynchronous setting with the synchronous version by evaluating the wall clock time required for the SO-AET-k to reach the final results obtained by SO-SP at the end of optimization.

The comparison is shown in the first row in Table 4.3 for each scenario, which represents the percentage of the wall clock time required by averaged results of SO-AET-k (denoted as $t_{\text{SO-AET-k}}$) that reach the averaged best result achieved by the synchronous setting SO-SP at 10000 seconds to the total 10000 seconds. It can be denoted as $\eta_{\text{SO-AET-k}}$ (i.e. $\eta_{\text{SO-AET-k}} = t_{\text{SO-AET-k}}/10000$). The less percentage $\eta_{\text{SO-AET-k}}$ is, the more efficient asynchronous setting SO-AET-k is.

The asynchronous version SO-AET-k only requires in general around 50% wall

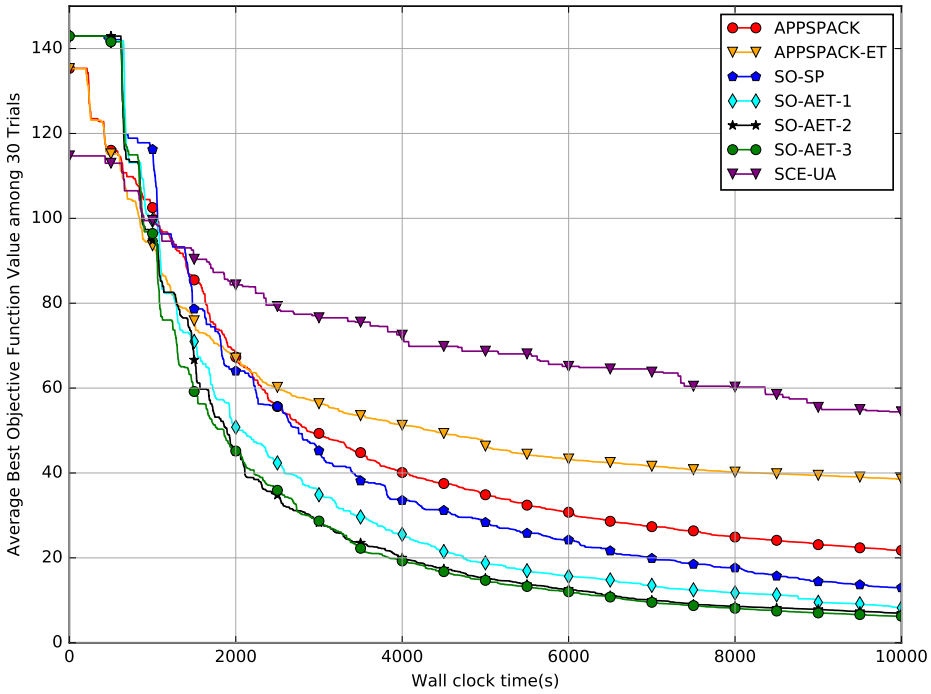


Figure 4.9: Progress Graph of results Scenario 2 which compare averaged best results among 30 trials found so far at each wall-clock time for different Rules knowledge extraction of SO-AET-k with its synchronous version SO-SP, and other two algorithms as SCE-UA, APPSPACK and APPSPACK-ET (i.e. APPSPACK with Early truncation defined same as in Section 4.4.3)

clock time to achieve the same solutions obtained by synchronous version SO-SP at 10000 seconds for both scenarios (Table 4.3). As to alternative algorithms, SO-AET-k needs around 30% of the time required by APPSPACK, 20% of the time required by APPSPACK-ET and less than 20% of what SCE-UA takes to obtain their results at 10000 seconds. And we can find $t_{\text{SO-AET-3}}$ is the most time efficient setting compared among all Rules of Knowledge Extraction for saving computation budget required by SO-SP as it obtains the smallest $\eta_{\text{SO-AET-k}}$ among SO-AET-k.

Table 4.3: The table of percentages of wall clock time t_A required for averaged results of algorithm A in column including different Rules of Knowledge Extraction (i.e. SO-AET-k for k=1,2,3) and SO-SP to reach the averaged results obtained from alternative algorithms B in row including SO-SP, APPSPACK, APPSPACK-ET after 10000 seconds among 30 trials (i.e. $t_A/10000$)

Scenario	Benchmark	SO-SP	SO-AET-1	SO-AET-2	SO-AET-3
1	SO-SP	-	50.12%	53.77%	40.96%
	APPSPACK	44.94%	34.97%	33.09%	27.03%
	APPSPACK-ET	33.58%	26.19%	24.93%	22.03%
	SCE-UA	19.39%	18.17%	18.22%	14.72%
2	SO-SP	-	71.49%	57.93%	55.28%
	APPSPACK	65%	44.42%	37.99%	35.41%
	APPSPACK-ET	34.97	27.82%	22.14%	23.23%
	SCE-UA	18.32%	15.35%	14.93%	13.03%

4.6.3 Stochastic Performance analysis

Due to the fact that both SO-SP and SO-AET-k are stochastic optimization algorithms, various trials may have different performances. Therefore, we need to analyze the consistency of the performance among different trials. We conducted a statistical test of the results achieved by synchronous version SO-SP to variants of asynchronous version SO-AET-k. In this study, Mann-Whitney Rank Sum Test is used as it is a non-parameter test which does not have any pre-assumption for the underlying distribution in the dataset. We use the simulation results after 10000 seconds of 30 trials for each setting for both SO-SP and SO-AET-k.

In Table 4.4, results of SO-SP show that it has no significant differences compared to results from APPSPACK and its variant because APPSPACK has high variance (Fig. 4.10), but it is significantly better than SCE-UA at 1% level. As to SO-AET-k, their results are statistically better than SO-SP at 10000 second in both scenarios, especially SO-AET-2 and SO-AET-3 for which they are both sig-

Table 4.4: P-values from statistical comparison of different SO-AET-k with SO-SP, APPSPACK, SCE-UA and SO-SP compared to APPSPACK, SCE-UA by Mann-Whitney Rank Sum Test after 10000 seconds. At a significance level of $\alpha\%$, a p-value $< \alpha\%$ means that algorithm A in column (such as SO-SP, SO-AET-k) is significantly better than algorithm B in row (such as SO-SP, APPSPACK, APPSPACK-ET and SCE-UA). (orange colored P-values indicate significantly different at 1% level, olive green colored P-values indicate significantly different at 5% level, dark green colored P-values indicate significantly different at 10% level, black colored P-values indicate no significant difference)

S	setting	SO-SP	SO-AET-1	SO-AET-2	SO-AET-3
1	SO-SP	1	8.93×10^{-5}	3.96×10^{-5}	6.28×10^{-7}
	APPSPACK	0.48	5.1×10^{-2}	7.78×10^{-3}	8.87×10^{-3}
	APPSPACK-ET	0.918	4.28×10^{-2}	7.78×10^{-3}	8.87×10^{-3}
	SCE-UA	1.12×10^{-5}	1.53×10^{-7}	4.58×10^{-6}	2.79×10^{-9}
2	SO-SP	1	2.92×10^{-4}	4.22×10^{-5}	6.97×10^{-6}
	APPSPACK	0.76	5.1×10^{-2}	1.01×10^{-2}	5.70×10^{-3}
	APPSPACK-ET	0.220	8.79×10^{-4}	2.60×10^{-4}	1.54×10^{-4}
	SCE-UA	2.87×10^{-11}	3.88×10^{-11}	2.87×10^{-11}	2.87×10^{-11}

nificantly better than all the algorithms compared at 1% level except for one case that SO-AET-2 is significantly better than APPSPACK in Scenario 2 at 5% level. And all SO-AET-k have a p-value small than 10% compared against APPSPACK, APPSPACK-ET and SCE-UA, indicating that all the comparisons illustrate that asynchronous settings are significantly better than SO-SP, APPSPACK and especially SCE-UA at 90% confidence level.

We plot the box plots of the simulation results of 30 trials for each of the settings for both SO-SP, SO-AET-k, SCE-UA, APPSPACK and APPSPACK-ET after 10000 seconds. In Figure 4.10 and Figure 4.11, it shows that results from all SO-AET-k with different Rules of Knowledge Extraction have smallest median as well as smallest range of values among all algorithms after 10000 seconds in both scenarios. In Scenario 1, compared with SO-SP, SO-AET-k have smaller outliers, and SO-AET-3 has the fewest outliers in SO-AET-k. And in Scenario 2,

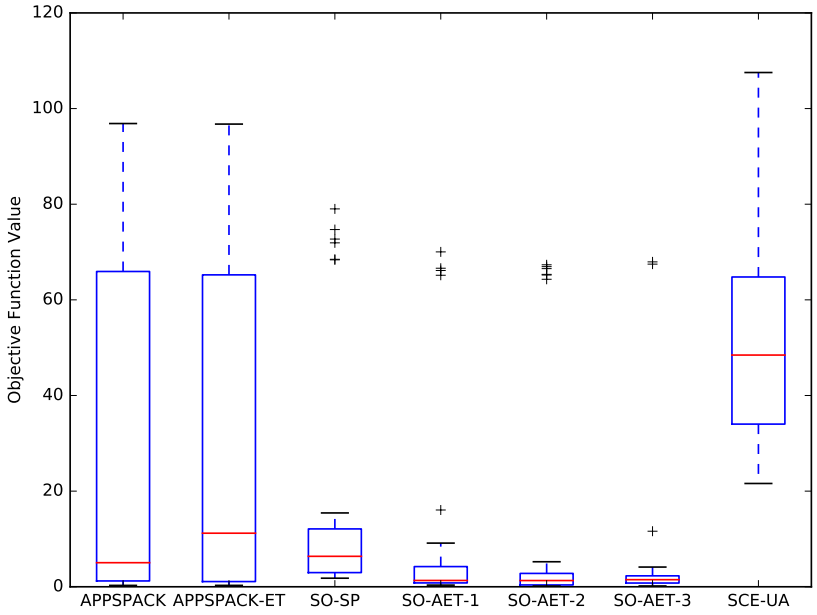


Figure 4.10: Box plot of results for Scenario 1 among 30 trails at 10000 seconds for different Rules knowledge extraction of SO-AET-k with its synchronous version SO-SP, and other two algorithms as SCE-UA, APPSPACK and APPSPACK with Early truncation defined same as in Section 4.4.3

SO-AET-3 has no outlier. The algorithm which has the largest value range are APPSPACK and its variant APPSPACK-ET in both scenarios. As APPSPACK is a local optimizer, it is reasonable to see this behavior, since certain trials in APPSPACK get trapped in some local optimum, the range of the results can be large. As a global optimizer, SCE-UA has a relatively small range of values. However, the median of results from SCE-UA is the highest among all algorithms, which indicates that SCE-UA has a consistently bad performance in both scenarios.

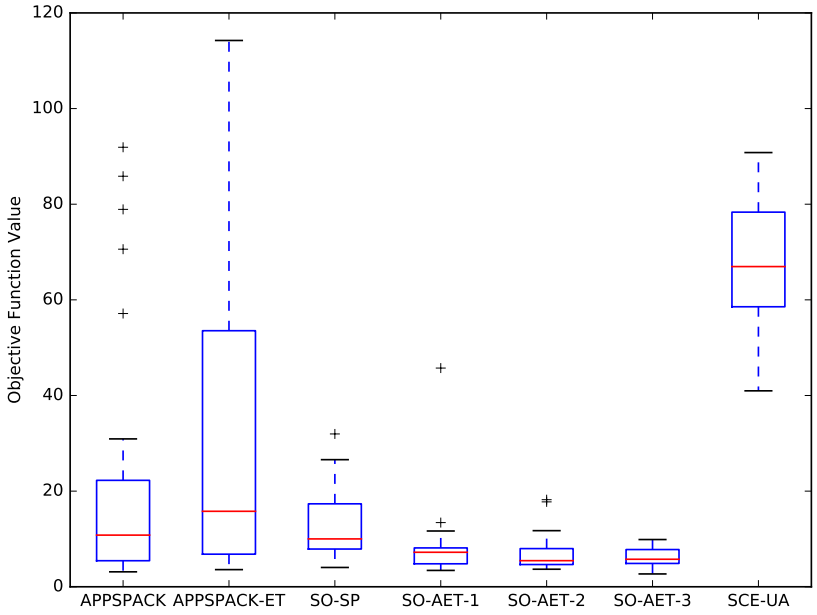


Figure 4.11: Box plot of results for Scenario 2 among 30 trails at 10000 seconds for different Rules knowledge extraction of SO-AET-k with its synchronous version SO-SP, and other two algorithms as SCE-UA, APPSPACK and APPSPACK with Early truncation defined same as in Section 4.4.3

4.7 Conclusion

In this study, we introduce early truncation strategy along with different Rules of Knowledge Extraction in asynchronous version of surrogate-based optimization algorithm (SO-AET-k) and find out that it favors searching the optimum. The coupled asynchronous parallel and the early truncation strategy has great advantage for calibration problem with objective as cumulative sum of squared errors. The empirical truncation strategy designed for this study introduce unbalanced load, and provides asynchronous paradigms great benefits over its synchronous version. As shown in this study, the solutions obtained by SO-AET-k are significantly better than SO-SP, and moreover computational time can be saved when

using asynchronous paradigm compared the synchronous setting for the problem in two scenarios, results shows that $40\% - 70\%$ of the computational time can be saved in Umatilla problems in order to obtain the same averaged results level of the synchronous version.

We also found that by incorporating all the information from truncated points (SO-AET-3) is beneficial, even though the truncation is based on estimation and their evaluated objective function cannot be directly used when coupled with the algorithm. However, for problems with varying trends of function values or less certainty in forecasting the function evaluations, not incorporating the evaluation information from truncation points maybe a favorable option (SO-AET-2), or we can also regard the truncated points as points we've never evaluated (SO-AET-1).

All in all, SO-AET show a consistently better performance compared to synchronous version (SO-SP) as well as two other algorithms compared SCE-UA and APPSPACK along with APPSPACK-ET in terms of computation saving, results quality and robustness in finding good optimal solution. For calibration of groundwater flow and transport model with expensive computation budget, the problem of having significant optimization time can be saved. Beyond that, this system is not only limited in implementation in the calibration process, but also suitable for problems which have a somewhat predictable objective values such as monotonically increasing or decreasing objective functions.

CHAPTER 5

CONCLUSION

Parallel Surrogate-based Optimization Algorithms are efficient and robust for solving computationally expensive groundwater problems. In this dissertation, analyses have been conducted to evaluate the performances of several Parallel Surrogate-based optimization algorithms. The algorithms include a well-developed Stochastic Radial Basis function algorithm in its parallel version, a combined Parallel DYCORS for high dimension problems with expensive constraint functions dealing strategy, and also a new algorithm developed for monotonically increasing function by using early termination strategy which is specifically suitable for parameter calibration problems with expensive models.

In Chapter 2, we measure the efficiency of Parallel RBF by using different computing resources (i.e. different number core counts) on Yellowstone supercomputer system in the application of two real world groundwater remediation problems. We used new metric which evaluate parallel algorithm efficiency by reaching the same accuracy level instead of the completing the same amount of function evaluations and what we have shown is that Parallel RBF can reach super-linear speed up, which in another words meaning its efficiency exceed 100% compared to the serial version when using a small number of cores. In addition, for large of number of cores in use, Parallel RBF largely reduce the computation time required for reaching the same results as using its serial version, which gives it one competitive advantage over using small number of cores. Moreover, compared to three other popular parallel global optimization algorithms i.e. GA, NOMAD and APPSPACK, parallel Stochastic RBF performs significantly and consistently better than the other three not only in its averaged performance but also in a

statistical way of evaluation.

Chapter 3 describes land subsidence problems with large number of dimensions (38 decision variables) in Hang-Jia-Hu area in China which involve a computationally expensive model integrating both groundwater flow and subsidence simulations. In order to work out plan for optimal groundwater extraction based on different extraction requirements, three different formulations are designed in consideration of groundwater demand, land subsidence control, and test on new aquifer exploitation. For constraints concerning the evaluation of the integrated land subsidence model, we introduce additional surrogate surfaces for constraints to facilitate the search of feasible domain. Results have shown that with the help of the additional surrogate surface, we are able to reduce the computational budget (i.e. number of function evaluations) for reaching the same optimal solution level both in averaged performance and in statistical level of comparison. Compared to popular parallel GA, the efficient performance of DYSOC is more prominent as it can reduce even more computational budget. Overall, in all of the three formulations, the new algorithm we introduced provide the efficient and consistent performance.

Studies in Chapter 2 and 3 are focusing on synchronous paradigm in parallelism of Surrogate-based optimization algorithms. In Chapter 4, we adapt the asynchronous paradigm of Parallel DYCORS and introduce a new early truncation strategy which is especially useful to parameter calibration problems which have monotonically increasing objective functions based on computationally expensive models. This new algorithm can efficiently solve for calibration problem as it both avoids evaluation on bad solutions and prevents cores for expensive evaluation being idle. Based on the concept, we implement the incorporated asyn-

chronous DYCORS with early termination strategy by using three different linkage policies. Our application are on two real-world based calibration problems with multiple local optima. The results have shown that the preservation of early truncated points could help the algorithm increase its efficiency for finding the optimal solution. Overall the results of comparison among different algorithms illustrate that the new incorporation strategy has the better performance compared to its synchronous version, synchronous parallel GA, as well as asynchronous algorithm APPSPACK. Also the statical comparison shows that new algorithm is significantly better than the other algorithms, which illustrate the robust performance of our new algorithm.

This dissertation has shown that Parallelism in Surrogate-based Optimization Algorithms provide increasing efficiency and can outperform many parallel global optimization algorithms in groundwater. New algorithm additional surrogate for constraint function and asynchronous paradigm with early truncation strategy can aid problems with specific characteristic and help improve the reduction of computational budget required to find the optimal solution. Further study can be also devoted to models with multiple characteristic, such as high dimensional problems with expensive constraint functions and semi-predictable objective functions so that an early truncation strategy can be useful. In addition, even though our application focused on managing groundwater issues and calibrating for groundwater systems, these algorithms can be further implemented into other computationally expensive physical based models.

BIBLIOGRAPHY

[1] M.A. Abramson, C. Audet, G. Couture, J.E. Dennis, Jr., S. Le Digabel, and C. Tribes. The NOMAD project. Software available at `https://www.gerad.ca/nomad/`.

[2] David P Ahlfeld and Gemma Baro-montes. Solving Unconfined Groundwater Flow management Problems with Successive Linear Programming. *Journal of Water Resources Planning and Management*, 134(5):404–413, 2008.

[3] William M Alley. Tracking U.S. Groundwater: Reserves for the Future? *Environment: Science and Policy for Sustainable Development*, 48(3):10–25, April 2006.

[4] C. Audet, S. Le Digabel, and C. Tribes. NOMAD user guide. Technical Report G-2009-37, Les cahiers du GERAD, 2009.

[5] M Björkman and K Holmström. Global Optimization of Costly Nonconvex Functions Using Radial Basis Functions. *Optimization and Engineering*, 1(4):373–397, 2000.

[6] A. I. Calderhead, R. Therrien, A. Rivera, R. Martel, and J. Garfias. Simulating pumping-induced regional land subsidence with the use of InSAR and field data in the Toluca Valley, Mexico. *Advances in Water Resources*, 34(1):83–97, 2011.

[7] Guoliang Cao, Dongmei Han, and Jessa Moser. Groundwater exploitation management under land subsidence constraint: Empirical evidence from the hangzhou-jiaxing-huzhou plain, China. *Environmental Management*, 51(6):1109–1125, 2013.

[8] Yin-lung Chang, Tung-lin Tsai, Jinn-chuang Yang, M Asce, and Yeou-koung Tung. Considering Land Subsidence. *Journal of Water Resources Planning and Management*, 133(December):486–498, 2007.

[9] Jen Min Cheng and William W G Yeh. A proposed quasi-Newton method for parameter identification in a flow and transport system. *Advances in Water Resources*, 15(4):239–249, 1992.

[10] Yung-Chia Chiu. Application of differential evolutionary optimization methodology for parameter structure identification in groundwater modeling. *Hydrogeology Journal*, 22(8):1731–1748, 2014.

[11] Zhenxue Dai. Inverse problem of multicomponent reactive chemical transport in porous media: Formulation and applications. *Water Resources Research*, 40:W07407, 2004.

[12] Larry M Deschaine, Theodore P Lillys, and János D Pintér. Groundwater remediation design using physics-based flow, transport, and optimization technologies. *Environmental Systems Research*, 2(1):1–21, 2013.

[13] Je Doherty and Rj Hunt. Approaches to highly parameterized inversion: a guide to using PEST for groundwater-model calibration. *U. S. Geological Survey Scientific Investigations Report 2010-5169*, page 70, 2010.

[14] Yanhui Dong, Guomin Li, and Haizhen Xu. Distributed Parallel Computing in Stochastic Modeling of Groundwater Systems. *Ground Water*, 51(2):no–no, July 2012.

[15] Qingyun Duan, Soroosh Sorooshian, and Vijai K. Gupta. Optimal use of the SCE-UA global optimization method for calibrating watershed models. *Journal of Hydrology*, 158(3-4):265–284, 1994.

[16] Y Duan. Shuffled Complex Evolution Approach for Effective and Efficient Global Minimization. *JOURNAL OF OPTIMIZATION THEORY AND APPLICATIONS*, 76(3), 1993.

[17] David Eriksson and David Bindel. pySOT: Surrogate Optimization Toolbox. `https://pypi.python.org/pypi/pySOT`.

[18] David Eriksson, David Bindel, and Christine A Shoemaker. *Surrogate Optimization Toolbox (pySOT)*, 2015.

[19] Félix-Antoine Fortin, François-Michel De Rainville, Marc-André Gardner, Marc Parizeau, and Christian Gagné. DEAP: Evolutionary algorithms made easy. *Journal of Machine Learning Research*, 13:2171–2175, jul 2012.

[20] Devin L. Galloway and Thomas J. Burbey. Review: Regional land subsidence accompanying groundwater extraction. *Hydrogeology Journal*, 19(8):1459–1486, 2011.

[21] Craig A Garrett, Junqi Huang, Mark N Goltz, and Gary B Lamont. Parallel real-valued genetic algorithms for bioremediation optimization of tce-contaminated groundwa - Evolutionary Computation, 1999. CEC 99. Proceedings of the 1999 Congress on. *IEEE*, pages 1–7, March 1999.

[22] Craig A. Garrett, Junqi Huang, Mark N. Goltz, and Gary B. Lamont. Parallel real-valued genetic algorithms for bioremediation optimization of TCE-contaminated groundwater. *Proceedings of the 1999 Congress on Evolutionary Computation, CEC 1999*, 3:2183–2189, 1999.

[23] Michel Gendreau. Parallel Asynchronous Tabu Search for Multicommodity Location-Allocation with Balancing Requirements Teodor Gabriel Crainic Michel Toulouse. 63(November):277–299, 1994.

[24] F. Giacobbo, M. Marseguerra, and E. Zio. Solving the inverse problem of parameter estimation by genetic algorithms: The case of a groundwater contaminant transport model. *Annals of Nuclear Energy*, 29(8):967–981, 2002.

[25] Genetha A. Gray and Tamara G. Kolda. APPSPACK 4.0: Asynchronous parallel pattern search for derivative-free optimization. Technical Report SAND2004-6391, Sandia National Laboratories, Livermore, CA 94551, August 2004.

[26] Haipeng Guo, Zuochen Zhang, Guoming Cheng, Wenpeng Li, Tiefeng Li, and Jiu Jimmy Jiao. Groundwater-derived land subsidence in the North China Plain. *Environmental Earth Sciences*, 74(2):1415–1427, 2015.

[27] Xiaoniu Guo, Chuan-Mian Zhang, and John C. Borthwick. Successive equimarginal approach for optimal design of a pump and treat system. *Water Resources Research*, 43(8):n/a–n/a, 2007. W08416.

[28] H.-M. Gutmann. A radial basis function method for global optimization. *Journal of Global Optimization*, 19(3):201–227, 03 2001. Copyright - Kluwer Academic Publishers 2001; Last updated - 2014-08-23.

[29] O. Bozorg Haddad, M. Mohammad Rezapour Tabari, E. Fallah-Mehdipour, and M. A. Mariño. Groundwater Model Calibration by Meta-Heuristic Algorithms. *Water Resources Management*, 27(7):2515–2529, 2013.

[30] A.W. Harbaugh. *MODFLOW-2005 , The U . S . Geological Survey Modular Ground-Water Model the Ground-Water Flow Process*. 2005.

[31] Harbaugh, A.W. MODFLOW-2005, The U.S. Geological Survey Modular Ground-Water Model—the Ground-Water Flow Process: U.S. Geological Survey Techniques and Methods. pages 6–A16, January 2005.

[32] Thomas Hemker, Kathleen R. Fowler, Matthew W. Farthing, and Oskar von

Stryk. A mixed-integer simulation-based optimization approach with surrogate functions in water resources management. *Optimization and Engineering*, 9(4):341–360, Dec 2008.

[33] M.C. Hill and C.R. Tiedeman. *Effective groundwater model calibration: With analysis of data, sensitivities, predictions, and uncertainty*, volume 46. 2007.

[34] Jörn Hoffmann, S.A. Leake, D.L. Galloway, and A.M Wilson. MODFLOW-2000 ground-water modelUser guide to the Subsidence and Aquifer-System Compaction (SUB) Package. Technical report, 2003.

[35] Patricia D. Hough, Tamara G. Kolda, and Virginia J. Torczon. Asynchronous Parallel Pattern Search for Nonlinear Optimization. *SIAM Journal on Scientific Computing*, 23(1):134–156, 2001.

[36] Tobias Houska, Philipp Kraft, Alejandro Chamorro-Chavez, and Lutz Breuer. SPOTting model parameters using a ready-made python package. *PLoS ONE*, 10(12), 2015.

[37] R. L. Hu, Z. Q. Yue, L. C. Wang, and S. J. Wang. Review on current status and challenging issues of land subsidence in China. *Engineering Geology*, 76(1-2):65–77, 2004.

[38] F. Huang, G. H. Wang, Y. Y. Yang, and C. B. Wang. Overexploitation status of groundwater and induced geological hazards in China. *Natural Hazards*, 73(2):727–741, 2014.

[39] Donald R Jones, Matthias Schonlau, and William J Welch. Efficient Global Optimization of Expensive Black-Box Functions. *Journal of Global Optimization*, 13:455–492, 1998.

[40] Ineke M. Kalwij and Richard C. Peralta. Simulation/optimization modeling for robust pumping strategy design. *Ground Water*, 44(4):574–582, 2006.

[41] Ineke M. Kalwij and Richard C. Peralta. Non-adaptive and adaptive hybrid approaches for enhancing water quality management. *Journal of Hydrology*, 358(34):182 – 192, 2008.

[42] Halil Karahan and M. Tamer Ayvaz. Simultaneous parameter identification of a heterogeneous aquifer system using artificial neural networks. *Hydrogeology Journal*, 16(5):817–827, 2008.

[43] Hamed Ketabchi and Behzad Ataie-Ashtiani. Assessment of a parallel evolutionary optimization approach for efficient management of coastal aquifers. *Environmental Modelling & Software*, 74:21–38, 2015.

[44] Byung Il Koh, Alan D. George, Raphael T. Haftka, and Benjamin J. Fregly. Parallel asynchronous particle swarm optimization. *International Journal for Numerical Methods in Engineering*, 67(4):578–595, 2006.

[45] Tamara G. Kolda. Revisiting asynchronous parallel pattern search. Technical Report SAND2004-8055, Sandia National Laboratories, Livermore, CA 94551, February 2004.

[46] K. J. Larson, H. Baaaolu, and M. A. Mariño. Prediction of optimal safe ground water yield and land subsidence in the Los Banos-Kettleman City area, California, using a calibrated numerical simulation model. *Journal of Hydrology*, 242(1-2):79–102, 2001.

[47] B Minsker, Y Zhang, R Greenwald, R Peralta, C Zheng, K Harre, D Becker, L Yeh, and K Yager. Final technical report for application of flow and transport optimization codes to groundwater pump and treat systems. *Environmental Security Technology, Certification Program, Arlington, Va*, 2003.

[48] Baha Y. Mirghani, Kumar G. Mahinthakumar, Michael E. Tryby, Ranji S. Ranjithan, and Emily M. Zechman. A parallel evolutionary strategy based simulation-optimization approach for solving groundwater source identification problems. *Advances in Water Resources*, 32(9):1373–1385, 2009.

[49] Baha Y Mirghani, Kumar G Mahinthakumar, Michael E Tryby, Ranji S Ranjithan, and Emily M Zechman. A parallel evolutionary strategy based simulation–optimization approach for solving groundwater source identification problems. *Advances in Water Resources*, 32(9):1373–1385, September 2009.

[50] Juliane Müller and Christine A. Shoemaker. Influence of ensemble surrogate models and sampling strategy on the solution quality of algorithms for??computationally expensive black-box global optimization problems. *Journal of Global Optimization*, 60(2):123–144, 2014.

[51] Richard C Peralta. *Groundwater optimization handbook : flow, contaminant transport, and conjunctive management*. Boca Raton, FL : Taylor & Francis, 2012.

[52] N. Phien-wej, P.H. Giao, and P. Nutalaya. Land subsidence in Bangkok, Thailand. *Engineering Geology*, 82(4):187–201, 2006.

[53] By Steven P Phillips, Carl S Carlson, Loren F Metzger, James F Howle, Devin L Galloway, Michelle Sneed, Marti E Ikehara, Kenneth W Hudnut, and Nancy E King. Analysis of tests of subsurface injection , storage , and recovery of freshwater in Lancaster , Antelope Valley , California. Technical report, 2003.

[54] Patrick M Reed and Joshua B Kollat. Visual analytics clarify the scalability and effectiveness of massively parallel many-objective optimization: A groundwater monitoring design example. *Advances in Water Resources*, 56:1–13, June 2013.

[55] R. G. Regis and C. A. Shoemaker. Constrained global optimization of expensive black box functions using radial basis functions. *J. Global Opt*, pages 31(1), 153–171, 2005.

[56] Rommel G. Regis. Stochastic radial basis function algorithms for large-scale optimization involving expensive black-box objective and constraint functions. *Computers and Operations Research*, 38(5):837–853, 2011.

[57] Rommel G. Regis and Christine A. Shoemaker. Improved strategies for radial basis function methods for global optimization. *Journal of Global Optimization*, 37(1):113–135, 2007.

[58] Rommel G Regis and Christine A Shoemaker. Parallel Stochastic Global Optimization Using Radial Basis Functions. *INFORMS Journal on Computing*, 21(3):411–426, July 2009.

[59] Rommel G. Regis and Christine A. Shoemaker. Parallel stochastic global optimization using radial basis functions. *INFORMS Journal on Computing*, 21(3):411–426, 2009.

[60] Rommel G. Regis and Christine a. Shoemaker. Combining radial basis function surrogates and dynamic coordinate search in high-dimensional expensive black-box optimization. *Engineering Optimization*, 45(5):529–555, 2013.

[61] M Sayeed and G K Mahinthakumar. Efficient parallel implementation of hybrid optimization approaches for solving groundwater inverse problems. *Journal of Computing in Civil Engineering*, 19(4):329–340, 2005.

[62] Mohamed Sayeed, G Kumar Mahinthakumar, and M ASCE. Efficient Parallel Implementation of Hybrid Optimization Approaches for Solving Groundwater Inverse Problems. *Journal of Computing in Civil Engineering*, pages 1–12, August 2005.

[63] Special Section. Global change and the groundwatermanagement challenge Steven. *Water Resources Research*, pages 3031–3051, 2015.

[64] Songqing Shan and G. Gary Wang. Survey of modeling and optimization strategies to solve high-dimensional design problems with computationally-expensive black-box functions. *Structural and Multidisciplinary Optimization*, 41(2):219–241, Mar 2010.

[65] Abhishek Singh and Barbara Minsker. Modeling and characterization of uncertainty for optimization of groundwater remediation at the umatilla chemical depot. In *American Society of Civil Engineers (ASCE) Environmental & Water Resources Institute (EWRI) World Water & Environmental Resources Congress 2003 & Related Symposia, Philadelphia, PA*, 2003.

[66] Amandeep Singh. *Efficient Optimization of Computationally Expensive Problems Using A New Parallel Algorithm And Response Surface Based Methods*. PhD thesis, Cornell University, 5 2011.

[67] Ashutosh Singh, Claudius M. Bürger, and Olaf A. Cirpka. Optimized Sustainable Groundwater Extraction Management: General Approach and Application to the City of Lucknow, India. *Water Resources Management*, 27(12):4349–4368, 2013.

[68] Eva Sinha and Barbara S. Minsker. Multiscale island injection genetic algorithms for groundwater remediation. *Advances in Water Resources*, 30(9):1933 – 1942, 2007.

[69] D P Solomatine, Y B Dibike, and N Kukuric. Automatic calibration of groundwater models using global optimization techniques. 44(6):879–894, 1999.

[70] P. Teatini, M. Ferronato, G. Gambolati, and M. Gonella. Groundwater pumping and land subsidence in the Emilia-Romagna coastland, Italy: Modeling the past occurrence and the future trend. *Water Resources Research*, 42(1):1–19, 2006.

[71] Bryan A. Tolson and Christine A. Shoemaker. Dynamically dimensioned search algorithm for computationally efficient watershed model calibration. *Water Resources Research*, 43(1):1–16, 2007.

[72] Christian von Lücken, Benjamin Barán, and Aldo Sotelo. Pump Scheduling Optimization Using Asynchronous Parallel Evolutionary Algorithms. *CLEI Electorinic Journal*, 7(2):1–20, 2004.

[73] Jasper A. Vrugt, Breanndán Ó Nualláin, Bruce A. Robinson, Willem Bouten, Stefan C. Dekker, and Peter M A Sloot. Application of parallel computing to stochastic parameter estimation in environmental models. *Computers and Geosciences*, 32(8):1139–1155, 2006.

[74] Q J Wang. The Genetic Algorithm and Its Application to Calibrating Conceptual Rainfall-Runoff Models. 27(9):2467–2471, 1991.

[75] Shengquan Yan and Barbara Minsker. Optimal groundwater remediation design using an Adaptive Neural Network Genetic Algorithm. *Water Resources Research*, 42(5):n/a–n/a, May 2006.

[76] Yun Yang, Jianfeng Wu, Xiaomin Sun, Jichun Wu, and Chunmiao Zheng. Development and application of a master-slave parallel hybrid multi-objective evolutionary algorithm for groundwater remediation design. *Environmental Earth Sciences*, 70(6):2481–2494, February 2013.

[77] Yan Zhang, Robert Greenwald, Barbara Minsker, Richard Peralta, and Chunmiao Zheng. Final cost and performance report application of flow and transport optimization codes to groundwater pump and treat systems. Technical report, DTIC Document, 2004.

[78] Chunmiao Zheng and P. Patrick Wang. MT3DMS: A Modular Three-Dimensional Multispecies Transport Model for simulation of advection, dispersion and chemical reactions of contaminants in groundwater systems. *A modular three-dimensional multi-species ...*, (December):239, 1999.

[79] Chunmiao Zheng, P. Patrick Wang, Chunmiao Zheng, and P. Patrick Wang. Mt3dms: A modular three-dimensional multi-species transport model for simulation of advection, dispersion, and chemical reactions of contaminants in ground-water systems. documentation and user's guide. In *Contract Report SERDP-99-1, U.S. Army Engineer Research and Development*, 1999.

[80] Chunmiao Zheng and Patrick Wang. Application of flow and transport optimization codes to groundwater pump-and-treat systems: Umatilla army depot, oregon. 2001.